Notice

This manual and any examples contained herein are provided “as is” and are subject to change without notice. Hewlett-Packard Company makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard Co. shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein.

© Copyright Hewlett-Packard Company 1993. All rights reserved. Reproduction, adaptation, or translation of this manual is prohibited without prior written permission of Hewlett-Packard Company, except as allowed under the copyright laws.

The programs that control this product are copyrighted and all rights are reserved. Reproduction, adaptation, or translation of those programs without prior written permission of Hewlett-Packard Co. is also prohibited.

© Trustees of Columbia University in the City of New York, 1989. Permission is granted to any individual or institution to use, copy, or redistribute Kermit software so long as it is not sold for profit, provided this notice is retained.

Hewlett-Packard Company
Corvallis Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Acknowledgements

Hewlett-Packard gratefully acknowledges the members of the Education Advisory Committee (Dr. Thomas Dick, Dr. Lynn Garner, Dr. John Kenelly, Dr. Don LaTorre, Dr. Jerold Mathews, and Dr. Gil Proctor) for their assistance in the development of this product. Special thanks are also due to Donald R. Asmus, Scott Burke, Bhushan Gupta and his students at the Oregon Institute of Technology, and Carla Randall and her AP Calculus students.

Edition History

Edition 1 ................................................................. July 1993
Edition 2 ................................................................. January 1994
Edition 3 ................................................................. May 1994
Edition 4 ................................................................. December 1994
1. Programming

Understanding Programming ................................................. 1-1
   The Contents of a Program ............................................. 1-1
   Calculations in a Program ............................................. 1-3
Entering and Executing Programs ........................................... 1-4
Viewing and Editing Programs ............................................. 1-9
Creating Programs on a Computer ......................................... 1-10
Using Local Variables ..................................................... 1-11
   Creating Local Variables .............................................. 1-11
   Evaluating Local Names .............................................. 1-13
   Defining the Scope of Local Variables .............................. 1-14
   Compiled Local Variables ............................................ 1-15
Creating User-Defined Functions as Programs ......................... 1-16
Using Tests and Conditional Structures ................................ 1-17
   Testing Conditions .................................................... 1-17
      Using Comparison Functions ...................................... 1-17
      Using Logical Functions ......................................... 1-19
      Testing Object Types ............................................. 1-20
      Testing Linear Structure ........................................ 1-20
Using Conditional Structures and Commands ............................ 1-20
   The IF ... THEN ... END Structure ................................. 1-20
   The IFT Command ...................................................... 1-21
   The IF ... THEN ... ELSE ... END Structure ...................... 1-21
   The IFTE Function .................................................... 1-22
   The CASE ... END Structure ......................................... 1-22
   Conditional Examples ................................................. 1-23
Using Loop Structures .................................................... 1-27
   Using Definite Loop Structures ...................................... 1-28
      The START ... NEXT Structure .................................. 1-28
      The START ... STEP Structure .................................. 1-30
   The FOR ... NEXT Structure ......................................... 1-32
   The FOR ... STEP Structure ......................................... 1-34
Using Indefinite Loop Structures ................................. 1-36
   The DO ... UNTIL ... END Structure ........................ 1-36
   The WHILE ... REPEAT ... END Structure .................... 1-38
Using Loop Counters .............................................. 1-39
Using Summations Instead of Loops .............................. 1-40
Using Flags .......................................................... 1-42
   Types of Flags .................................................. 1-42
   Setting, Clearing, and Testing Flags .......................... 1-42
   Recalling and Storing the Flag States ....................... 1-44
Using Subroutines .................................................. 1-45
Single-Stepping through a Program ............................... 1-47
Trapping Errors .................................................... 1-50
   Causing and Analyzing Errors ................................. 1-51
   Making an Error Trap ......................................... 1-53
   The IFERR ... THEN ... END Structure ....................... 1-53
   The IFERR ... THEN ... ELSE ... END Structure .......... 1-54
Input ........................................................................... 1-55
   Data Input Commands ............................................. 1-56
   Using PROMPT ... CONT for Input .............................. 1-56
   Using DISP FREEZE HALT ... CONT for Input ............... 1-58
   Using INPUT ... ENTER for Input ............................. 1-60
   Using INFORM and CHOOSE for Input ......................... 1-67
   Beeping to Get Attention ...................................... 1-71
Stopping a Program for Keystroke Input ......................... 1-72
   Using WAIT for Keystroke Input ............................... 1-72
   Using KEY for Keystroke Input ............................... 1-73
Output ........................................................................ 1-74
   Data Output Commands ......................................... 1-74
   Labeling Output with Tags ..................................... 1-74
   Labeling and Displaying Output as Strings ................ 1-75
   Pausing to Display Output .................................... 1-76
   Using MSGBOX to Display Output .............................. 1-77
Using Menus with Programs ........................................ 1-77
   Using Menus for Input .......................................... 1-79
   Using Menus to Run Programs ................................. 1-79
Turning Off the HP 48 from a Program ......................... 1-82
2. Programming Examples

Fibonacci Numbers ........................................... 2-2
  FIB1 (Fibonacci Numbers, Recursive Version) ........ 2-2
  FIB2 (Fibonacci Numbers, Loop Version) .............. 2-3
  FIBT (Comparing Program-Execution Time) ............... 2-5
Displaying a Binary Integer .............................. 2-7
  PAD (Pad with Leading Spaces) ............................ 2-7
  PRESERVE (Save and Restore Previous Status) ........ 2-8
BDISP (Binary Display) .................................... 2-10
Median of Statistics Data ................................. 2-14
  %TILE (Percentile of a List) ............................ 2-14
  MEDIAN (Median of Statistics Data) .................... 2-16
Expanding and Collecting Completely ..................... 2-19
  MULTI (Multiple Execution) .............................. 2-19
  EXCO (Expand and Collect Completely) ................. 2-20
Minimum and Maximum Array Elements ..................... 2-22
  MNX (Minimum or Maximum Element—Version 1) ....... 2-22
  MNX2 (Minimum or Maximum Element—Version 2) ....... 2-25
Applying a Program to an Array .......................... 2-29
Converting Between Number Bases ........................ 2-32
Verifying Program Arguments .............................. 2-36
  NAMES (Check List for Exactly Two Names) ............ 2-36
  VFY (Verify Program Argument) .......................... 2-38
Converting Procedures from Algebraic to RPN ............. 2-40
Bessel Functions ........................................... 2-43
Animation of Successive Taylor’s Polynomials .......... 2-45
  SINTP (Converting a Plot to a Graphics Object) .... 2-45
  SETTS (Superimposing Taylor’s Polynomials) ......... 2-46
  TSA (Animating Taylor’s Polynomials) ................. 2-47
Programmatic Use of Statistics and Plotting .......... 2-49
Trace Mode .................................................. 2-53
Inverse-Function Solver .................................. 2-54
Animating a Graphical Image ............................. 2-56

3. Command Reference

ABS ......................................................... 3-5
ACK ......................................................... 3-6
ACKALL ..................................................... 3-6
ACOS ......................................................... 3-7
ACOSH ....................................................... 3-9
ADD ......................................................... 3-11
ALOG .................................................. 3-12
AMORT ............................................... 3-12
AND .................................................. 3-13
ANIMATE ............................................. 3-14
APPLY ............................................... 3-15
ARC .................................................. 3-17
ARCHIVE ........................................... 3-18
ARG .................................................. 3-19
ARRY→ ............................................. 3-19
ARRY ............................................... 3-20
ASIN ............................................... 3-21
ASINH .............................................. 3-23
ASN .................................................. 3-24
ASR .................................................. 3-25
ATAN ............................................... 3-26
ATANH .............................................. 3-28
ATICK .............................................. 3-30
ATTACH ............................................ 3-31
AUTO ............................................... 3-32
AXES ............................................... 3-33
BAR .................................................. 3-34
BARPLOT .......................................... 3-36
BAUD ............................................... 3-36
BEEP ............................................... 3-37
BESTFIT .......................................... 3-37
BIN .................................................. 3-38
BINS ............................................... 3-38
BLANK ............................................ 3-39
BOX .................................................. 3-40
BUFLLEN ........................................... 3-40
BYTES ............................................. 3-41
B→R .................................................. 3-42
CASE ............................................... 3-42
CEIL ............................................... 3-44
CENTR ............................................ 3-44
CF .................................................... 3-45
CHOOSE .......................................... 3-46
%CH ............................................... 3-47
CHR ............................................... 3-48
CKSM .............................................. 3-49
CLEAR ............................................. 3-50
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKADJ</td>
<td>3-50</td>
</tr>
<tr>
<td>CLLCD</td>
<td>3-51</td>
</tr>
<tr>
<td>CLOSEIO</td>
<td>3-51</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>3-52</td>
</tr>
<tr>
<td>CLTEACH</td>
<td>3-52</td>
</tr>
<tr>
<td>CLUSR</td>
<td>3-52</td>
</tr>
<tr>
<td>CLVAR</td>
<td>3-53</td>
</tr>
<tr>
<td>CNRM</td>
<td>3-53</td>
</tr>
<tr>
<td>→COL</td>
<td>3-54</td>
</tr>
<tr>
<td>COL+</td>
<td>3-54</td>
</tr>
<tr>
<td>COL−</td>
<td>3-55</td>
</tr>
<tr>
<td>COL→</td>
<td>3-56</td>
</tr>
<tr>
<td>COLCT</td>
<td>3-56</td>
</tr>
<tr>
<td>COLΣ</td>
<td>3-57</td>
</tr>
<tr>
<td>COMB</td>
<td>3-58</td>
</tr>
<tr>
<td>CON</td>
<td>3-59</td>
</tr>
<tr>
<td>COND</td>
<td>3-60</td>
</tr>
<tr>
<td>CONIC</td>
<td>3-61</td>
</tr>
<tr>
<td>CONJ</td>
<td>3-62</td>
</tr>
<tr>
<td>CONLIB</td>
<td>3-63</td>
</tr>
<tr>
<td>CONST</td>
<td>3-63</td>
</tr>
<tr>
<td>CONT</td>
<td>3-64</td>
</tr>
<tr>
<td>CONVERT</td>
<td>3-65</td>
</tr>
<tr>
<td>CORR</td>
<td>3-65</td>
</tr>
<tr>
<td>COS</td>
<td>3-66</td>
</tr>
<tr>
<td>COSH</td>
<td>3-67</td>
</tr>
<tr>
<td>COV</td>
<td>3-67</td>
</tr>
<tr>
<td>CR</td>
<td>3-68</td>
</tr>
<tr>
<td>CRDIR</td>
<td>3-69</td>
</tr>
<tr>
<td>CROSS</td>
<td>3-69</td>
</tr>
<tr>
<td>CSWP</td>
<td>3-70</td>
</tr>
<tr>
<td>CYLIN</td>
<td>3-70</td>
</tr>
<tr>
<td>C→PX</td>
<td>3-71</td>
</tr>
<tr>
<td>C→R</td>
<td>3-71</td>
</tr>
<tr>
<td>DARCY</td>
<td>3-72</td>
</tr>
<tr>
<td>DATE</td>
<td>3-73</td>
</tr>
<tr>
<td>→DATE</td>
<td>3-73</td>
</tr>
<tr>
<td>DATE+</td>
<td>3-74</td>
</tr>
<tr>
<td>DEBUG</td>
<td>3-74</td>
</tr>
<tr>
<td>DDAYS</td>
<td>3-75</td>
</tr>
<tr>
<td>DEC</td>
<td>3-75</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>DECR</td>
<td>3-76</td>
</tr>
<tr>
<td>DEFINE</td>
<td>3-77</td>
</tr>
<tr>
<td>DEG</td>
<td>3-78</td>
</tr>
<tr>
<td>DELALARM</td>
<td>3-78</td>
</tr>
<tr>
<td>DELAY</td>
<td>3-79</td>
</tr>
<tr>
<td>DELKEYS</td>
<td>3-80</td>
</tr>
<tr>
<td>DEPND</td>
<td>3-81</td>
</tr>
<tr>
<td>DEPTH</td>
<td>3-82</td>
</tr>
<tr>
<td>DET</td>
<td>3-82</td>
</tr>
<tr>
<td>DETACH</td>
<td>3-84</td>
</tr>
<tr>
<td>DIAG→</td>
<td>3-84</td>
</tr>
<tr>
<td>→DIAG</td>
<td>3-85</td>
</tr>
<tr>
<td>DIFFEQ</td>
<td>3-86</td>
</tr>
<tr>
<td>DISP</td>
<td>3-88</td>
</tr>
<tr>
<td>DO</td>
<td>3-89</td>
</tr>
<tr>
<td>DOERR</td>
<td>3-90</td>
</tr>
<tr>
<td>DOLIST</td>
<td>3-91</td>
</tr>
<tr>
<td>DOSUBS</td>
<td>3-92</td>
</tr>
<tr>
<td>DOT</td>
<td>3-94</td>
</tr>
<tr>
<td>DRAW</td>
<td>3-94</td>
</tr>
<tr>
<td>DRAX</td>
<td>3-95</td>
</tr>
<tr>
<td>DROP</td>
<td>3-95</td>
</tr>
<tr>
<td>DROPN</td>
<td>3-96</td>
</tr>
<tr>
<td>DROP2</td>
<td>3-96</td>
</tr>
<tr>
<td>DTAG</td>
<td>3-97</td>
</tr>
<tr>
<td>DUP</td>
<td>3-97</td>
</tr>
<tr>
<td>DUPN</td>
<td>3-98</td>
</tr>
<tr>
<td>DUP2</td>
<td>3-98</td>
</tr>
<tr>
<td>D→R</td>
<td>3-99</td>
</tr>
<tr>
<td>EGV</td>
<td>3-100</td>
</tr>
<tr>
<td>EGVL</td>
<td>3-101</td>
</tr>
<tr>
<td>ELSE</td>
<td>3-101</td>
</tr>
<tr>
<td>END</td>
<td>3-102</td>
</tr>
<tr>
<td>ENDSUB</td>
<td>3-102</td>
</tr>
<tr>
<td>ENG</td>
<td>3-103</td>
</tr>
<tr>
<td>EQ→</td>
<td>3-104</td>
</tr>
<tr>
<td>EQNLIB</td>
<td>3-104</td>
</tr>
<tr>
<td>ERASE</td>
<td>3-105</td>
</tr>
<tr>
<td>ERRM</td>
<td>3-105</td>
</tr>
<tr>
<td>ERRN</td>
<td>3-106</td>
</tr>
<tr>
<td>Term</td>
<td>Page</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>ERR0</td>
<td>3-106</td>
</tr>
<tr>
<td>EVAL</td>
<td>3-107</td>
</tr>
<tr>
<td>EXP</td>
<td>3-108</td>
</tr>
<tr>
<td>EXPAN</td>
<td>3-109</td>
</tr>
<tr>
<td>EXPFIT</td>
<td>3-110</td>
</tr>
<tr>
<td>EXPM</td>
<td>3-110</td>
</tr>
<tr>
<td>EYEPT</td>
<td>3-111</td>
</tr>
<tr>
<td>F0λ</td>
<td>3-112</td>
</tr>
<tr>
<td>FACT</td>
<td>3-112</td>
</tr>
<tr>
<td>FANNING</td>
<td>3-113</td>
</tr>
<tr>
<td>FC?·</td>
<td>3-114</td>
</tr>
<tr>
<td>FC?C</td>
<td>3-114</td>
</tr>
<tr>
<td>FFT</td>
<td>3-115</td>
</tr>
<tr>
<td>FINDALARM</td>
<td>3-116</td>
</tr>
<tr>
<td>FINISH</td>
<td>3-117</td>
</tr>
<tr>
<td>FIX</td>
<td>3-117</td>
</tr>
<tr>
<td>FLOOR</td>
<td>3-118</td>
</tr>
<tr>
<td>FOR</td>
<td>3-119</td>
</tr>
<tr>
<td>FP</td>
<td>3-121</td>
</tr>
<tr>
<td>FREE</td>
<td>3-121</td>
</tr>
<tr>
<td>FREE1</td>
<td>3-122</td>
</tr>
<tr>
<td>FREEZE</td>
<td>3-123</td>
</tr>
<tr>
<td>FS?</td>
<td>3-124</td>
</tr>
<tr>
<td>FS?C</td>
<td>3-125</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>3-125</td>
</tr>
<tr>
<td>GET</td>
<td>3-127</td>
</tr>
<tr>
<td>GETI</td>
<td>3-129</td>
</tr>
<tr>
<td>GOR</td>
<td>3-130</td>
</tr>
<tr>
<td>GRAD</td>
<td>3-131</td>
</tr>
<tr>
<td>GRAPH</td>
<td>3-131</td>
</tr>
<tr>
<td>GRIDMAP</td>
<td>3-132</td>
</tr>
<tr>
<td>-&gt;GROB</td>
<td>3-133</td>
</tr>
<tr>
<td>GXOR</td>
<td>3-134</td>
</tr>
<tr>
<td>*H</td>
<td>3-135</td>
</tr>
<tr>
<td>HALT</td>
<td>3-136</td>
</tr>
<tr>
<td>HEAD</td>
<td>3-136</td>
</tr>
<tr>
<td>HEX</td>
<td>3-137</td>
</tr>
<tr>
<td>HISTOGRAM</td>
<td>3-137</td>
</tr>
<tr>
<td>HISTPLOT</td>
<td>3-139</td>
</tr>
<tr>
<td>HMS+</td>
<td>3-139</td>
</tr>
<tr>
<td>HMS−</td>
<td>3-140</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>HMS</td>
<td>3-141</td>
</tr>
<tr>
<td>→HMS</td>
<td>3-142</td>
</tr>
<tr>
<td>HOME</td>
<td>3-142</td>
</tr>
<tr>
<td>i</td>
<td>3-143</td>
</tr>
<tr>
<td>IDN</td>
<td>3-143</td>
</tr>
<tr>
<td>IF</td>
<td>3-144</td>
</tr>
<tr>
<td>IFERR</td>
<td>3-146</td>
</tr>
<tr>
<td>IFFT</td>
<td>3-147</td>
</tr>
<tr>
<td>IFT</td>
<td>3-148</td>
</tr>
<tr>
<td>IFTE</td>
<td>3-149</td>
</tr>
<tr>
<td>IM</td>
<td>3-150</td>
</tr>
<tr>
<td>INCR</td>
<td>3-150</td>
</tr>
<tr>
<td>INDEP</td>
<td>3-151</td>
</tr>
<tr>
<td>INFORM</td>
<td>3-152</td>
</tr>
<tr>
<td>INPUT</td>
<td>3-154</td>
</tr>
<tr>
<td>INV</td>
<td>3-156</td>
</tr>
<tr>
<td>IP</td>
<td>3-157</td>
</tr>
<tr>
<td>IR</td>
<td>3-157</td>
</tr>
<tr>
<td>ISOL</td>
<td>3-158</td>
</tr>
<tr>
<td>KERRM</td>
<td>3-159</td>
</tr>
<tr>
<td>KEY</td>
<td>3-159</td>
</tr>
<tr>
<td>KGET</td>
<td>3-160</td>
</tr>
<tr>
<td>KILL</td>
<td>3-161</td>
</tr>
<tr>
<td>LABEL</td>
<td>3-162</td>
</tr>
<tr>
<td>LAST</td>
<td>3-162</td>
</tr>
<tr>
<td>LASTARG</td>
<td>3-163</td>
</tr>
<tr>
<td>LCD</td>
<td>3-163</td>
</tr>
<tr>
<td>→LCD</td>
<td>3-164</td>
</tr>
<tr>
<td>LIBEVAL</td>
<td>3-165</td>
</tr>
<tr>
<td>LIBS</td>
<td>3-165</td>
</tr>
<tr>
<td>LINE</td>
<td>3-166</td>
</tr>
<tr>
<td>ΣLINE</td>
<td>3-166</td>
</tr>
<tr>
<td>LINFIT</td>
<td>3-167</td>
</tr>
<tr>
<td>LININ</td>
<td>3-168</td>
</tr>
<tr>
<td>LIST</td>
<td>3-169</td>
</tr>
<tr>
<td>→LIST</td>
<td>3-169</td>
</tr>
<tr>
<td>ΣLIST</td>
<td>3-170</td>
</tr>
<tr>
<td>ΔLIST</td>
<td>3-170</td>
</tr>
<tr>
<td>ΠLIST</td>
<td>3-171</td>
</tr>
<tr>
<td>LN</td>
<td>3-172</td>
</tr>
<tr>
<td>LNP1</td>
<td>3-174</td>
</tr>
</tbody>
</table>
LOG 3-174
LOGFIT 3-176
LQ 3-176
LR 3-177
LSQ 3-178
LU 3-179
MANT 3-179
MATCH 3-180
\textup{\textasciitilde}MATCH 3-181
MAX 3-183
MAXR 3-183
MAX\Sigma 3-184
MCALC 3-185
MEAN 3-185
MEM 3-186
MENU 3-187
MERGE 3-190
MERGE1 3-190
MIN 3-191
MIN\textup{\textasciitilde}HUNT 3-192
MINIT 3-193
MINR 3-193
MIN\Sigma 3-194
MITM 3-194
MOD 3-195
MROOT 3-195
MSGBOX 3-196
MSOLVR 3-197
MUSER 3-197
NDIST 3-198
NEG 3-199
NEWOB 3-200
NEXT 3-201
NEXT 3-201
NOT 3-201
NOVAL 3-203
NSUB 3-203
NUM 3-204
\textasciitilde NUM 3-207
NUMX 3-207
NUMY 3-208
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σ</td>
<td>3-208</td>
</tr>
<tr>
<td>OBJ→</td>
<td>3-209</td>
</tr>
<tr>
<td>OCT</td>
<td>3-210</td>
</tr>
<tr>
<td>OFF</td>
<td>3-211</td>
</tr>
<tr>
<td>OLDPRT</td>
<td>3-211</td>
</tr>
<tr>
<td>OPENIO</td>
<td>3-212</td>
</tr>
<tr>
<td>OR</td>
<td>3-213</td>
</tr>
<tr>
<td>ORDER</td>
<td>3-214</td>
</tr>
<tr>
<td>OVER</td>
<td>3-215</td>
</tr>
<tr>
<td>PARAMETRIC</td>
<td>3-215</td>
</tr>
<tr>
<td>PARITY</td>
<td>3-217</td>
</tr>
<tr>
<td>PARSURFACE</td>
<td>3-218</td>
</tr>
<tr>
<td>PATH</td>
<td>3-219</td>
</tr>
<tr>
<td>PCOEF</td>
<td>3-220</td>
</tr>
<tr>
<td>PCONTOUR</td>
<td>3-221</td>
</tr>
<tr>
<td>PCOV</td>
<td>3-222</td>
</tr>
<tr>
<td>PDIM</td>
<td>3-223</td>
</tr>
<tr>
<td>PERM</td>
<td>3-224</td>
</tr>
<tr>
<td>PEVAL</td>
<td>3-224</td>
</tr>
<tr>
<td>PGDIR</td>
<td>3-225</td>
</tr>
<tr>
<td>PICK</td>
<td>3-226</td>
</tr>
<tr>
<td>PICT</td>
<td>3-226</td>
</tr>
<tr>
<td>PICTURE</td>
<td>3-227</td>
</tr>
<tr>
<td>PINIT</td>
<td>3-228</td>
</tr>
<tr>
<td>PILOF</td>
<td>3-228</td>
</tr>
<tr>
<td>PIXON</td>
<td>3-229</td>
</tr>
<tr>
<td>PIX?</td>
<td>3-229</td>
</tr>
<tr>
<td>PKT</td>
<td>3-230</td>
</tr>
<tr>
<td>PMAX</td>
<td>3-231</td>
</tr>
<tr>
<td>PMIN</td>
<td>3-231</td>
</tr>
<tr>
<td>POLAR</td>
<td>3-232</td>
</tr>
<tr>
<td>POS</td>
<td>3-234</td>
</tr>
<tr>
<td>PREDV</td>
<td>3-234</td>
</tr>
<tr>
<td>PREDX</td>
<td>3-235</td>
</tr>
<tr>
<td>PREDY</td>
<td>3-236</td>
</tr>
<tr>
<td>PRLCD</td>
<td>3-237</td>
</tr>
<tr>
<td>PROMPT</td>
<td>3-238</td>
</tr>
<tr>
<td>PROOT</td>
<td>3-238</td>
</tr>
<tr>
<td>PRST</td>
<td>3-239</td>
</tr>
<tr>
<td>PRSTC</td>
<td>3-240</td>
</tr>
<tr>
<td>PRVAR</td>
<td>3-240</td>
</tr>
</tbody>
</table>

Contents-10
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR1</td>
<td>3-241</td>
</tr>
<tr>
<td>PSDEV</td>
<td>3-242</td>
</tr>
<tr>
<td>PURGE</td>
<td>3-243</td>
</tr>
<tr>
<td>PUT</td>
<td>3-244</td>
</tr>
<tr>
<td>PUTI</td>
<td>3-246</td>
</tr>
<tr>
<td>PVAR</td>
<td>3-247</td>
</tr>
<tr>
<td>PVARS</td>
<td>3-248</td>
</tr>
<tr>
<td>PVVIEW</td>
<td>3-249</td>
</tr>
<tr>
<td>PWRFIT</td>
<td>3-250</td>
</tr>
<tr>
<td>PX→C</td>
<td>3-250</td>
</tr>
<tr>
<td>→Q</td>
<td>3-251</td>
</tr>
<tr>
<td>→Qπ</td>
<td>3-251</td>
</tr>
<tr>
<td>QR</td>
<td>3-253</td>
</tr>
<tr>
<td>QUAD</td>
<td>3-253</td>
</tr>
<tr>
<td>QUOTE</td>
<td>3-254</td>
</tr>
<tr>
<td>RAD</td>
<td>3-256</td>
</tr>
<tr>
<td>RAND</td>
<td>3-256</td>
</tr>
<tr>
<td>RANK</td>
<td>3-257</td>
</tr>
<tr>
<td>RANM</td>
<td>3-257</td>
</tr>
<tr>
<td>RATIO</td>
<td>3-258</td>
</tr>
<tr>
<td>RCEQ</td>
<td>3-259</td>
</tr>
<tr>
<td>RCI</td>
<td>3-260</td>
</tr>
<tr>
<td>RCIJ</td>
<td>3-260</td>
</tr>
<tr>
<td>RCL</td>
<td>3-261</td>
</tr>
<tr>
<td>RCLALARM</td>
<td>3-262</td>
</tr>
<tr>
<td>RCLF</td>
<td>3-262</td>
</tr>
<tr>
<td>RCLKEYS</td>
<td>3-263</td>
</tr>
<tr>
<td>RCLMENU</td>
<td>3-264</td>
</tr>
<tr>
<td>RCLΣ</td>
<td>3-264</td>
</tr>
<tr>
<td>RCWS</td>
<td>3-265</td>
</tr>
<tr>
<td>RDM</td>
<td>3-266</td>
</tr>
<tr>
<td>RDZ</td>
<td>3-267</td>
</tr>
<tr>
<td>RE</td>
<td>3-267</td>
</tr>
<tr>
<td>RECN</td>
<td>3-268</td>
</tr>
<tr>
<td>RECT</td>
<td>3-269</td>
</tr>
<tr>
<td>RECV</td>
<td>3-269</td>
</tr>
<tr>
<td>REPEAT</td>
<td>3-270</td>
</tr>
<tr>
<td>REPL</td>
<td>3-270</td>
</tr>
<tr>
<td>RES</td>
<td>3-271</td>
</tr>
<tr>
<td>RESTORE</td>
<td>3-273</td>
</tr>
<tr>
<td>REVLIST</td>
<td>3-274</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>RKF</td>
<td>3-274</td>
</tr>
<tr>
<td>RKFERR</td>
<td>3-276</td>
</tr>
<tr>
<td>RKFSTEP</td>
<td>3-277</td>
</tr>
<tr>
<td>RL</td>
<td>3-278</td>
</tr>
<tr>
<td>RLB</td>
<td>3-278</td>
</tr>
<tr>
<td>RND</td>
<td>3-279</td>
</tr>
<tr>
<td>RNRM</td>
<td>3-280</td>
</tr>
<tr>
<td>ROLL</td>
<td>3-280</td>
</tr>
<tr>
<td>ROLLD</td>
<td>3-281</td>
</tr>
<tr>
<td>ROOT</td>
<td>3-281</td>
</tr>
<tr>
<td>ROT</td>
<td>3-282</td>
</tr>
<tr>
<td>ROW</td>
<td>3-282</td>
</tr>
<tr>
<td>ROW+</td>
<td>3-283</td>
</tr>
<tr>
<td>ROW−</td>
<td>3-284</td>
</tr>
<tr>
<td>ROW→</td>
<td>3-284</td>
</tr>
<tr>
<td>RR</td>
<td>3-285</td>
</tr>
<tr>
<td>RRB</td>
<td>3-285</td>
</tr>
<tr>
<td>RREF</td>
<td>3-286</td>
</tr>
<tr>
<td>RRK</td>
<td>3-286</td>
</tr>
<tr>
<td>RRKSTEP</td>
<td>3-288</td>
</tr>
<tr>
<td>RSBERR</td>
<td>3-290</td>
</tr>
<tr>
<td>RSD</td>
<td>3-291</td>
</tr>
<tr>
<td>RSWP</td>
<td>3-292</td>
</tr>
<tr>
<td>R—B</td>
<td>3-292</td>
</tr>
<tr>
<td>R—C</td>
<td>3-293</td>
</tr>
<tr>
<td>R—D</td>
<td>3-294</td>
</tr>
<tr>
<td>SAME</td>
<td>3-294</td>
</tr>
<tr>
<td>SBRK</td>
<td>3-295</td>
</tr>
<tr>
<td>SCALE</td>
<td>3-295</td>
</tr>
<tr>
<td>SCATR PLOT</td>
<td>3-296</td>
</tr>
<tr>
<td>SCATTER</td>
<td>3-297</td>
</tr>
<tr>
<td>SCHUR</td>
<td>3-298</td>
</tr>
<tr>
<td>SCI</td>
<td>3-299</td>
</tr>
<tr>
<td>SCLΣ</td>
<td>3-299</td>
</tr>
<tr>
<td>SCONJ</td>
<td>3-300</td>
</tr>
<tr>
<td>SDEV</td>
<td>3-300</td>
</tr>
<tr>
<td>SEND</td>
<td>3-301</td>
</tr>
<tr>
<td>SEQ</td>
<td>3-302</td>
</tr>
<tr>
<td>SERVER</td>
<td>3-303</td>
</tr>
<tr>
<td>SF</td>
<td>3-304</td>
</tr>
<tr>
<td>SHOW</td>
<td>3-305</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>SIDENS</td>
<td>3-305</td>
</tr>
<tr>
<td>SIGN</td>
<td>3-306</td>
</tr>
<tr>
<td>SIMU</td>
<td>3-307</td>
</tr>
<tr>
<td>SIN</td>
<td>3-307</td>
</tr>
<tr>
<td>SINH</td>
<td>3-308</td>
</tr>
<tr>
<td>SINV</td>
<td>3-309</td>
</tr>
<tr>
<td>SIZE</td>
<td>3-309</td>
</tr>
<tr>
<td>SL</td>
<td>3-310</td>
</tr>
<tr>
<td>SLB</td>
<td>3-311</td>
</tr>
<tr>
<td>SLOPEFIELD</td>
<td>3-312</td>
</tr>
<tr>
<td>SNEG</td>
<td>3-313</td>
</tr>
<tr>
<td>SNRM</td>
<td>3-314</td>
</tr>
<tr>
<td>SOLVEQN</td>
<td>3-314</td>
</tr>
<tr>
<td>SORT</td>
<td>3-315</td>
</tr>
<tr>
<td>SPHERE</td>
<td>3-316</td>
</tr>
<tr>
<td>SQ</td>
<td>3-316</td>
</tr>
<tr>
<td>SR</td>
<td>3-317</td>
</tr>
<tr>
<td>SRAD</td>
<td>3-317</td>
</tr>
<tr>
<td>SRB</td>
<td>3-318</td>
</tr>
<tr>
<td>SRECV</td>
<td>3-318</td>
</tr>
<tr>
<td>SST</td>
<td>3-320</td>
</tr>
<tr>
<td>SST</td>
<td>3-320</td>
</tr>
<tr>
<td>START</td>
<td>3-321</td>
</tr>
<tr>
<td>STD</td>
<td>3-322</td>
</tr>
<tr>
<td>STEP</td>
<td>3-323</td>
</tr>
<tr>
<td>STEQ</td>
<td>3-324</td>
</tr>
<tr>
<td>STIME</td>
<td>3-324</td>
</tr>
<tr>
<td>STO</td>
<td>3-325</td>
</tr>
<tr>
<td>STOALARM</td>
<td>3-326</td>
</tr>
<tr>
<td>STOF</td>
<td>3-327</td>
</tr>
<tr>
<td>STOKEYS</td>
<td>3-328</td>
</tr>
<tr>
<td>STO+</td>
<td>3-329</td>
</tr>
<tr>
<td>STO−</td>
<td>3-330</td>
</tr>
<tr>
<td>STO*</td>
<td>3-330</td>
</tr>
<tr>
<td>STO/</td>
<td>3-331</td>
</tr>
<tr>
<td>STOΣ</td>
<td>3-332</td>
</tr>
<tr>
<td>STREAM</td>
<td>3-333</td>
</tr>
<tr>
<td>STR→</td>
<td>3-333</td>
</tr>
<tr>
<td>→STR</td>
<td>3-334</td>
</tr>
<tr>
<td>STWS</td>
<td>3-335</td>
</tr>
<tr>
<td>SUB</td>
<td>3-336</td>
</tr>
</tbody>
</table>
4. Equation Reference

Columns and Beams (1) ........................................ 4-1
  Elastic Buckling (1, 1) .................................. 4-3
  Eccentric Columns (1, 2) .............................. 4-3
  Simple Deflection (1, 3) ............................... 4-4
  Simple Slope (1, 4) .................................... 4-5
  Simple Moment (1, 5) ................................... 4-6
  Simple Shear (1, 6) ..................................... 4-6
  Cantilever Deflection (1, 7) ......................... 4-7
  Cantilever Slope (1, 8) ................................ 4-7
  Cantilever Moment (1, 9) ............................... 4-8
  Cantilever Shear (1, 10) ............................... 4-9

Electricity (2) .............................................. 4-9
  Coulomb’s Law (2, 1) .................................. 4-11
  Ohm’s Law and Power (2, 2) ......................... 4-11
  Voltage Divider (2, 3) .................................. 4-12
  Current Divider (2, 4) .................................. 4-12
  Wire Resistance (2, 5) .................................. 4-13
  Series and Parallel R (2, 6) ......................... 4-13
  Series and Parallel C (2, 7) ......................... 4-14
  Series and Parallel L (2, 8) ......................... 4-14
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacitive Energy</td>
<td>4-15</td>
</tr>
<tr>
<td>Inductive Energy</td>
<td>4-15</td>
</tr>
<tr>
<td>RLC Current Delay</td>
<td>4-16</td>
</tr>
<tr>
<td>DC Capacitor Current</td>
<td>4-16</td>
</tr>
<tr>
<td>Capacitor Charge</td>
<td>4-17</td>
</tr>
<tr>
<td>DC Inductor Voltage</td>
<td>4-17</td>
</tr>
<tr>
<td>RC Transient</td>
<td>4-18</td>
</tr>
<tr>
<td>RL Transient</td>
<td>4-18</td>
</tr>
<tr>
<td>Resonant Frequency</td>
<td>4-19</td>
</tr>
<tr>
<td>Plate Capacitor</td>
<td>4-19</td>
</tr>
<tr>
<td>Cylindrical Capacitor</td>
<td>4-20</td>
</tr>
<tr>
<td>Solenoid Inductance</td>
<td>4-20</td>
</tr>
<tr>
<td>Toroid Inductance</td>
<td>4-21</td>
</tr>
<tr>
<td>Sinusoidal Voltage</td>
<td>4-21</td>
</tr>
<tr>
<td>Sinusoidal Current</td>
<td>4-21</td>
</tr>
<tr>
<td>Fluids</td>
<td>4-22</td>
</tr>
<tr>
<td>Pressure at Depth</td>
<td>4-23</td>
</tr>
<tr>
<td>Bernoulli Equation</td>
<td>4-23</td>
</tr>
<tr>
<td>Flow with Losses</td>
<td>4-24</td>
</tr>
<tr>
<td>Flow in Full Pipes</td>
<td>4-26</td>
</tr>
<tr>
<td>Forces and Energy</td>
<td>4-27</td>
</tr>
<tr>
<td>Linear Mechanics</td>
<td>4-28</td>
</tr>
<tr>
<td>Angular Mechanics</td>
<td>4-29</td>
</tr>
<tr>
<td>Centripetal Force</td>
<td>4-29</td>
</tr>
<tr>
<td>Hooke’s Law</td>
<td>4-30</td>
</tr>
<tr>
<td>1D Elastic Collisions</td>
<td>4-30</td>
</tr>
<tr>
<td>Drag Force</td>
<td>4-31</td>
</tr>
<tr>
<td>Law of Gravitation</td>
<td>4-31</td>
</tr>
<tr>
<td>Mass-Energy Relation</td>
<td>4-31</td>
</tr>
<tr>
<td>Gases</td>
<td>4-32</td>
</tr>
<tr>
<td>Ideal Gas Law</td>
<td>4-33</td>
</tr>
<tr>
<td>Ideal Gas State Change</td>
<td>4-33</td>
</tr>
<tr>
<td>Isothermal Expansion</td>
<td>4-34</td>
</tr>
<tr>
<td>Polytropic Processes</td>
<td>4-34</td>
</tr>
<tr>
<td>Isentropic Flow</td>
<td>4-35</td>
</tr>
<tr>
<td>Real Gas Law</td>
<td>4-36</td>
</tr>
<tr>
<td>Real Gas State Change</td>
<td>4-36</td>
</tr>
<tr>
<td>Kinetic Theory</td>
<td>4-37</td>
</tr>
<tr>
<td>Heat Transfer</td>
<td>4-37</td>
</tr>
<tr>
<td>Heat Capacity</td>
<td>4-38</td>
</tr>
<tr>
<td>Thermal Expansion</td>
<td>4-39</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Conduction (6, 3)</td>
<td>4-39</td>
</tr>
<tr>
<td>Convection (6, 4)</td>
<td>4-40</td>
</tr>
<tr>
<td>Conduction + Convection (6, 5)</td>
<td>4-41</td>
</tr>
<tr>
<td>Black Body Radiation (6, 6)</td>
<td>4-42</td>
</tr>
<tr>
<td>Magnetism (7)</td>
<td>4-43</td>
</tr>
<tr>
<td>Straight Wire (7, 1)</td>
<td>4-43</td>
</tr>
<tr>
<td>Force between Wires (7, 2)</td>
<td>4-44</td>
</tr>
<tr>
<td>Magnetic (B) Field in Solenoid (7, 3)</td>
<td>4-44</td>
</tr>
<tr>
<td>Magnetic (B) Field in Toroid (7, 4)</td>
<td>4-45</td>
</tr>
<tr>
<td>Motion (8)</td>
<td>4-46</td>
</tr>
<tr>
<td>Linear Motion (8, 1)</td>
<td>4-47</td>
</tr>
<tr>
<td>Object in Free Fall (8, 2)</td>
<td>4-47</td>
</tr>
<tr>
<td>Projectile Motion (8, 3)</td>
<td>4-48</td>
</tr>
<tr>
<td>Angular Motion (8, 4)</td>
<td>4-48</td>
</tr>
<tr>
<td>Circular Motion (8, 5)</td>
<td>4-49</td>
</tr>
<tr>
<td>Terminal Velocity (8, 6)</td>
<td>4-49</td>
</tr>
<tr>
<td>Escape Velocity (8, 7)</td>
<td>4-49</td>
</tr>
<tr>
<td>Optics (9)</td>
<td>4-50</td>
</tr>
<tr>
<td>Law of Refraction (9, 1)</td>
<td>4-51</td>
</tr>
<tr>
<td>Critical Angle (9, 2)</td>
<td>4-51</td>
</tr>
<tr>
<td>Brewster’s Law (9, 3)</td>
<td>4-52</td>
</tr>
<tr>
<td>Spherical Reflection (9, 4)</td>
<td>4-52</td>
</tr>
<tr>
<td>Spherical Refraction (9, 5)</td>
<td>4-53</td>
</tr>
<tr>
<td>Thin Lens (9, 6)</td>
<td>4-54</td>
</tr>
<tr>
<td>Oscillations (10)</td>
<td>4-54</td>
</tr>
<tr>
<td>Mass-Spring System (10, 1)</td>
<td>4-55</td>
</tr>
<tr>
<td>Simple Pendulum (10, 2)</td>
<td>4-56</td>
</tr>
<tr>
<td>Conical Pendulum (10, 3)</td>
<td>4-56</td>
</tr>
<tr>
<td>Torsional Pendulum (10, 4)</td>
<td>4-57</td>
</tr>
<tr>
<td>Simple Harmonic (10, 5)</td>
<td>4-57</td>
</tr>
<tr>
<td>Plane Geometry (11)</td>
<td>4-58</td>
</tr>
<tr>
<td>Circle (11, 1)</td>
<td>4-59</td>
</tr>
<tr>
<td>Ellipse (11, 2)</td>
<td>4-59</td>
</tr>
<tr>
<td>Rectangle (11, 3)</td>
<td>4-60</td>
</tr>
<tr>
<td>Regular Polygon (11, 4)</td>
<td>4-61</td>
</tr>
<tr>
<td>Circular Ring (11, 5)</td>
<td>4-61</td>
</tr>
<tr>
<td>Triangle (11, 6)</td>
<td>4-62</td>
</tr>
<tr>
<td>Solid Geometry (12)</td>
<td>4-63</td>
</tr>
<tr>
<td>Cone (12, 1)</td>
<td>4-64</td>
</tr>
<tr>
<td>Cylinder (12, 2)</td>
<td>4-64</td>
</tr>
<tr>
<td>Parallelepiped (12, 3)</td>
<td>4-65</td>
</tr>
</tbody>
</table>
Sphere (12, 4) ........................................... 4-66
Solid State Devices (13) ................................. 4-67
PN Step Junctions (13, 1) ............................... 4-69
NMOS Transistors (13, 2) ............................... 4-71
Bipolar Transistors (13, 3) ............................. 4-73
JFETs (13, 4) .............................................. 4-74
Stress Analysis (14) ...................................... 4-76
Normal Stress (14, 1) .................................. 4-77
Shear Stress (14, 2) .................................... 4-77
Stress on an Element (14, 3) ........................... 4-78
Mohr’s Circle (14, 4) .................................... 4-79
Waves (15) .................................................. 4-80
Transverse Waves (15, 1) ............................ 4-80
Longitudinal Waves (15, 2) ........................... 4-81
Sound Waves (15, 3) .................................... 4-81
References .................................................. 4-82

A.  Error and Status Messages

B.  Table of Units

C.  System Flags

D.  Reserved Variables

Contents of the Reserved Variables ........................ D-2
   ALRMDAT ................................................. D-2
   CST .................................................... D-3
   “der-” Names ......................................... D-4
   EQ ...................................................... D-4
   EXPR ................................................... D-5
   IOPAR .................................................. D-5
   Mpar .................................................... D-7
   Mpar .................................................... D-7
   n1, n2, .............................................. D-7
   Nmines ................................................ D-8
   PPAR .................................................. D-8
   PRTPAR ................................................. D-11
   s1, s2, ............................................ D-12
   VPAR .................................................. D-13
   ZPAR .................................................. D-14
   ΣDAT .................................................. D-15
   ΣPAR .................................................. D-16

Contents-19
E. New Commands

F. Technical Reference

Object Sizes .................................................. F-2
Automatic Simplification Rules ............................. F-3
Symbolic Integration Patterns ............................... F-5
Trigonometric Expansions ................................. F-7
Source References .......................................... F-9

G. Parallel Processing with Lists

Index
If you've used a calculator or computer before, you're probably familiar with the idea of programs. Generally speaking, a program is something that gets the calculator or computer to do certain tasks for you—more than a built-in command might do. In the HP 48, a program is an object that does the same thing.

Understanding Programming

An HP 48 program is an object with « » delimiters containing a sequence of numbers, commands, and other objects you want to execute automatically to perform a task.

For example, a program that takes a number from that stack, finds its factorial, and divides the result by 2 would look like this: «! 2 / » or

```
  «
  1
  2
  /
  »
```

The Contents of a Program

As mentioned above, a program contains a sequence of objects. As each object is processed in a program, the action depends on the type of object, as summarized below.
### Actions for Certain Objects in Programs

<table>
<thead>
<tr>
<th>Object</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Executed.</td>
</tr>
<tr>
<td>Number</td>
<td>Put on the stack.</td>
</tr>
<tr>
<td>Algebraic</td>
<td>Put on the stack.</td>
</tr>
<tr>
<td>String</td>
<td>Put on the stack.</td>
</tr>
<tr>
<td>List</td>
<td>Put on the stack.</td>
</tr>
<tr>
<td>Program</td>
<td>Put on the stack.</td>
</tr>
<tr>
<td>Global name (quoted)</td>
<td>Put on the stack.</td>
</tr>
<tr>
<td>Global name (unquoted)</td>
<td>Program executed.</td>
</tr>
<tr>
<td></td>
<td>Name evaluated.</td>
</tr>
<tr>
<td></td>
<td>Directory becomes current.</td>
</tr>
<tr>
<td></td>
<td>Other object put on the stack.</td>
</tr>
<tr>
<td>Local name (quoted)</td>
<td>Put on the stack.</td>
</tr>
<tr>
<td>Local name (unquoted)</td>
<td>Contents put on the stack.</td>
</tr>
</tbody>
</table>

As you can see from this table, most types of objects are simply put on the stack—but built-in commands and programs called by name cause *execution*. The following examples show the results of executing programs containing different sequences of objects.

#### Examples of Program Actions

<table>
<thead>
<tr>
<th>Program</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>«12»</td>
<td>2: 1</td>
</tr>
<tr>
<td></td>
<td>1: 2</td>
</tr>
<tr>
<td>«&quot;Hello&quot; {A B} »</td>
<td>2: &quot;Hello&quot;</td>
</tr>
<tr>
<td></td>
<td>1: {A B}</td>
</tr>
<tr>
<td>«'1+2' »</td>
<td>1: '1+2'</td>
</tr>
<tr>
<td>«'1+2' » NUM »</td>
<td>1: 3</td>
</tr>
<tr>
<td>««12+»»</td>
<td>1: «12+»</td>
</tr>
<tr>
<td>««12+» EVAL »</td>
<td>1: 3</td>
</tr>
</tbody>
</table>
Programs can also contain structures. A structure is a program segment with a defined organization. Two basic kinds of structures are available:

- **Local variable structure.** The → command defines local variable names and a corresponding algebraic or program object that’s evaluated using those variables.

- **Branching structures.** Structure words (like DO ... UNTIL ...) define conditional or loop structures to control the order of execution within a program.

A *local variable structure* has one of the following organizations inside a program:

```
→ name₁ ... nameₙ 'algebraic' »
→ name₁ ... nameₙ « program »
```

The → command removes \( n \) objects from the stack and stores them in the named local variables. The algebraic or program object in the structure is *automatically evaluated* because it’s an element of the structure—even though algebraic and program objects are put on the stack in other situations. Each time a local variable name appears in the algebraic or program object, the variable’s contents are substituted.

So the following program takes two numbers from the stack and returns a numeric result:

```
→ a b 'ABS(a-b)' »
```

**Calculations in a Program**

Many calculations in programs take data from the stack. Two typical ways to manipulate stack data are:

- **Stack commands.** Operate directly on the objects on the stack.

- **Local variable structures.** Stores the stack objects in temporary local variables, then uses the variable names to represent the data in the following algebraic or program object.

Numeric calculations provide convenient examples of these methods. The following programs use two numbers from the stack to calculate the hypotenuse of a right triangle using the formula \( \sqrt{x^2 + y^2} \).
The first program uses stack commands to manipulate the numbers on the stack—the calculation uses stack syntax. The second program uses a local variable structure to store and retrieve the numbers—the calculation uses stack syntax. The third program also uses a local variable structure—the calculation uses algebraic syntax. Note that the underlying formula is most apparent in the third program. This third method is often the easiest to write, read, and debug.

### Entering and Executing Programs

A program is an object—it occupies one level on the stack, and you can store it in a variable.

**To enter a program:**

1. Press \( \text{STO} \) \( \text{\textit{»}} \). The \( \text{PRG} \) annunciator appears, indicating Program-entry mode is active.
2. Enter the commands and other objects (with appropriate delimiters) in order for the operations you want the program to execute.
   - Press \( \text{SPC} \) to separate consecutive numbers.
   - Press \( \text{\textit{»}} \) to move past closing delimiters.
3. Optional: Press \( \text{\textit{»}} \) \( \text{(newline)} \) to start a new line in the command line at any time.
4. Press \( \text{\textit{»}} \) \( \text{ENTER} \) to put the program on the stack.

In Program-entry mode (\( \text{PRG} \) annunciator on), command keys aren’t executed—they’re entered in the command line instead. Only nonprogrammable operations such as \( \text{\textit{\textbullet}} \) and \( \text{VAR} \) are executed. Line breaks are discarded when you press \( \text{ENTER} \).

**To enter commands and other objects in a program:**

- Press the keyboard or menu key for the command or object.
  - or
- Type the characters using the alpha keyboard.

1-4 Programming
To store or name a program:
1. Enter the program on the stack.
2. Enter the variable name (with ' delimiters) and press (STO).

You can choose descriptive names for programs. Here are some ideas of what the name can describe:
- The calculation or action. Examples: SPH (spherical-cap volume), SORT (sort a list).
- The input and output. Examples: X→FX (x to f(x)), RH→V (radius-and-height to volume).
- The technique. Example: SPHLV (spherical-cap volume using local variables).

To execute a program:
- Press (VAR) then the menu key for the program name.
  or
- Enter the program name (with no delimiters) and press (ENTER).
  or
- Put the program name in level 1 and press (EVAL).
  or
- Put the program object in level 1 and press (EVAL).

To stop an executing program:
- Press (CANCEL).

Example: Enter a program that takes a radius value from the stack and calculates the volume of a sphere of radius $r$ using

$$V = \frac{4}{3} \pi r^3$$

If you were going to calculate the volume manually after entering the radius on the stack, you might press these keys:

\[ 3 \text{ ENTER} \pi \times 4 \text{ ENTER} 3 \div \times \text{ NUM} \]
Enter the same keystrokes in a program. (→ ←) just starts a new line.

```
3 π x 4
```

Put the program on the stack.

```
ENTER
```

Store the program in variable VOL. Then put a radius of 4 on the stack and run the VOL program.

```
VOL STO
4 VAR VOL
```

The program is

```
3 ^ π * 4 3 / * →NUM »
```

Example: Replace the program from the previous example with one that’s easier to read. Enter a program that uses a local variable structure to calculate the volume of a sphere. The program is

```
→ r '4/3*π*r^3' →NUM »
```

(You need to include →NUM because π causes a symbolic result.)

Enter the program. (→ ← just starts a new line.)

```
4 π x 3
```

Put the program on the stack, store it in VOL, and calculate the volume for a radius of 4.

```
4 VOL
```

1-6 Programming
**Example:** Enter a program $SPH$ that calculates the volume of a spherical cap of height $h$ within a sphere of radius $R$ using values stored in variables $H$ and $R$.

\[ V = \frac{1}{3} \pi h^2 (3r - h) \]

In this and following chapters on programming, “stack diagrams” show what arguments must be on the stack before a program is executed and what results the program leaves on the stack. Here’s the stack diagram for $SPH$.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>volume</td>
</tr>
</tbody>
</table>

The diagram indicates that $SPH$ takes no arguments from the stack and returns the volume of the spherical cap to level 1. ($SPH$ assumes that you’ve stored the numerical value for the radius in variable $R$ and the numerical value for the height in variable $H$. These are *global* variables—they exist outside the program.)

Program listings are shown with program steps in the left column and associated comments in the right column. Remember, you can either press the command keys or type in the command names to key in the program. In this first listing, the keystrokes are also shown.
### Program:
```
\( \langle \ \langle 1 \div 3 \times \pi \times h^2 \times (3 \times r - h)' \rangle \rightarrow \text{NUM} \rangle \)
```

### Keys:
```
\( \langle 1 \div 3 \times \pi \times h^2 \times (3 \times r - h)' \rangle \rightarrow \text{NUM} \langle \ \rangle \)
```

### Comments:
- **Begins the program.**
- **Begins the algebraic expression to calculate the volume.**
- **Multiplies by \( \pi h^2 \).**
- **Multiplies by \( 3r - h \), completing the calculation and ending the expression.**
- **Converts the expression with \( \pi \) to a number.**
- **Ends the program.**
- **Puts the program on the stack.**
- **Stores the program in variable \( SPH \).**

This is the program:
```
\( \langle \ \langle 1 \div 3 \times \pi \times h^2 \times (3 \times r - h)' \rangle \rightarrow \text{NUM} \rangle \)
```

Now use \( SPH \) to calculate the volume of a spherical cap of radius \( r = 10 \) and height \( h = 3 \).

First, store the data in the appropriate variables. Then select the VAR menu and execute the program. The answer is returned to level 1 of the stack.

```
10 \( \langle \ \rangle \times \ R \right \) \( \text{STO} \)
3 \( \langle \ \rangle \times \ H \right \) \( \text{STO} \)
\( \text{VAR} \rightarrow \text{SPH} \)
```

![1: 254.469004942](image)
Viewing and Editing Programs

You view and edit programs the same way you view and edit other objects—using the command line.

To view or edit a program:

1. View the program:
   - If the program is in level 1, press (EDIT) or (V).
   - If the program is stored in a variable, use the Memory Browser (MEMORY) to view the variable, or press (VAR) (ENTRY) and the variable’s menu key, followed by (V).

2. Optional: Make changes.

3. Press (ENTER) to save any changes (or press (CANCEL) to discard changes) and return to the stack.

The Memory Browser lets you change a stored program without having to do a store operation. (EDIT) lets you change a program and then store the new version in a different variable.

While you’re editing a program, you may want to switch the command-line entry mode between Program-entry mode (for editing most objects) and Algebraic/Program-entry mode (for editing algebraic objects). The PRG and ALG annunciators indicate the current mode.

To switch between entry modes:

Press (ENTRY).

Example: Edit \( SPH \) from the previous example so that it stores the number from level 1 into variable \( H \) and the number from level 2 into variable \( R \).

Use EDIT to start editing \( SPH \).

\[
\text{VAR} \quad \text{SPH} \quad \text{ENTRY}
\]

\[
\begin{align*}
\text{\( '1/3 \times \pi \times H^2 \times (3 \times R - H) \)} \quad & \rightarrow \text{NUM} \\
\end{align*}
\]

Programming 1-9
Move the cursor past the first program delimiter and insert the new program steps.

\[
\begin{align*}
\text{\texttt{H \ STO \ 'R' \ STO '1/3...}} \\
\text{\texttt{R \ STO}} \\
\end{align*}
\]

Save the edited version of \textit{SPH} in the variable. Then, to verify that the changes were saved, view \textit{SPH} in the command line.

\[
\begin{align*}
\text{\texttt{ENTER \ VAR \ \leftarrow \ SPH}} \\
\text{\texttt{SPH \ \downarrow}} \\
\end{align*}
\]

Press \texttt{(CANCEL)} to stop viewing.

---

**Creating Programs on a Computer**

It is convenient to create programs and other objects on a computer and then load them into the HP 48 using the calculator’s serial port.

If you are creating programs on a computer, you can include “comments” in the computer version of the program.

**To include a comment in a program:**

- Enclose the comment text between two @ characters.
- or
- Enclose the comment text between one @ character and the end of the line.

Whenever the HP 48 processes text entered in the command line—either from keyboard entry or transferred from a computer—it strips away the @ characters and the text they surround. However, @ characters are not affected if they’re inside a string.
Using Local Variables

The program *SPH* in the previous example uses global variables for data storage and recall. There are disadvantages to using global variables in programs:

- After program execution, global variables that you no longer need to use must be purged if you want to clear the VAR menu and free user memory.
- You must explicitly store data in global variables prior to program execution, or have the program execute STO.

Local variables address the disadvantages of global variables in programs. Local variables are temporary variables *created by a program*. They exist only while the program is being executed and cannot be used outside the program. They never appear in the VAR menu. In addition, local variables are accessed faster than global variables. (By convention, this manual uses lowercase names for local variables.) A compiled local variable is a form of local variable that can be used outside of the program that creates it. See “Compiled Local Variables” on page 1-15 for more information.

Creating Local Variables

In a program, a local variable structure creates local variables.

To enter a local variable structure in a program:

1. Enter the → command (press **(→)**).
2. Enter one or more variable names.
3. Enter a defining procedure (an algebraic or program object) that uses the names.

```
« → name_1 name_2 ... name_n 'algebraic' »
```

or

```
« → name_1 name_2 ... name_n « program » »
```

When the → command is executed in a program, *n* values are taken from the stack and assigned to variables *name_1, name_2, ... name_n*. For example, if the stack looks like this:
then

→ a creates local variable \( a = 20 \).

→ a \( \& b \) creates local variables \( a = 6 \) and \( b = 20 \).

→ a \( \& b \& c \) creates local variables \( a = 10 \), \( b = 6 \), and \( c = 20 \).

The defining procedure then uses the local variables to do calculations.

Local variable structures have these advantages:

- The → command stores the values from the stack in the corresponding variables—you don’t need to explicitly execute STO.

- Local variables automatically disappear when the defining procedure for which they are created has completed execution. Consequently, local variables don’t appear in the VAR menu, and they occupy user memory only during program execution.

- Local variables exist only within their defining procedure—different local variable structures can use the same variable names without conflict.

**Example:** The following program `SPHLV` calculates the volume of a spherical cap using local variables. The defining procedure is an algebraic expression.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>→</td>
<td>volume</td>
</tr>
</tbody>
</table>
Program:

```
*  → r h

'1/3ππh^2*(3*r-h)'

→NUM
```

Comments:

Creates local variables $r$ and $h$ for the radius of the sphere and height of the cap.
Expresses the defining procedure. In this program, the defining procedure for the local variable structure is an algebraic expression.
Converts expression to a number.
Stores the program in variable $SPHLV$.

Now use $SPHLV$ to calculate the volume of a spherical cap of radius $r = 10$ and height $h = 3$. Enter the data on the stack in the correct order, then execute the program.

```
10 ENTER 3 VAR SPHLV
```

Evaluating Local Names

Local names are evaluated differently from global names. When a global name is evaluated, the object stored in the corresponding variable is itself evaluated. (You’ve seen how programs stored in global variables are automatically evaluated when the name is evaluated.)

When a local name is evaluated, the object stored in the corresponding variable is returned to the stack but is not evaluated. When a local variable contains a number, the effect is identical to evaluation of a global name, since putting a number on the stack is equivalent to evaluating it. However, if a local variable contains a program, algebraic expression, or global variable name—and if you want it evaluated—the program should execute EVAL after the object is put on the stack.
Defining the Scope of Local Variables

Local variables exist only inside the defining procedure.

Example: The following program excerpt illustrates the availability of local variables in nested defining procedures (procedures within procedures). Because local variables \(a\), \(b\), and \(c\) already exist when the defining procedure for local variables \(d\), \(e\), and \(f\) is executed, they're available for use in that procedure.

Program:
```
+ a b c
- a b + c + 
+ d e f 
' a/(d*e+f)' 
- a c / -
```

Comments:
- No local variables are available.
- Defines local variables \(a\), \(b\), \(c\).
- Local variables \(a\), \(b\), \(c\) are available in this procedure.
- Defines local variables \(d\), \(e\), \(f\).
- Local variables \(a\), \(b\), \(c\) and \(d\), \(e\), \(f\) are available in this procedure.
- Only local variables \(a\), \(b\), \(c\) are available.
- No local variables are available.

Example: In the following program excerpt, the defining procedure for local variables \(d\), \(e\), and \(f\) calls a program that you previously created and stored in global variable \(P1\).
Program:

```
<
...
> a b c
<
  a b + c +
> d e f
'P1+a/(d*e+f)'
```

```
  a c / -
```

Comments:

Defines local variables \(d, e, f\).
Local variables \(a, b, c\) and \(d, e, f\) are available in this procedure. The defining procedure executes the program stored in variable \(P1\).

The six local variables are not available in program \(P1\) because they didn’t exist when you created \(P1\). The objects stored in the local variables are available to program \(P1\) only if you put those objects on the stack for \(P1\) to use or store those objects in global variables.

Conversely, program \(P1\) can create its own local variable structure (with any names, such as \(a, c,\) and \(f\), for example) without conflicting with the local variables of the same name in the procedure that calls \(P1\). It is possible to create a special type of local variable that can be used in other programs or subroutines. This type of local variable is called a compiled local variable.

**Compiled Local Variables**

Global variables use up memory, and local variables can’t be used outside of the program they were created in. Compiled local variables bridge the gap between these two variable types. To programs, compiled local variables look like global variables, but to the calculator they act like local variables. This means you can create a compiled local variable in a local variable structure, use it in any other program that is called within that structure, and when the program finishes, the variable is gone.
Compiled local variables have a special naming convention: they must begin with a \$. For example,

\[
\begin{align*}
\times & \\\\\rightarrow \\\\\\$u \\
' \text{IFTE(}$u<0\text{, BELOW, ABOVE)'} & \\
\times
\end{align*}
\]

The variable \$u is a compiled local variable that can be used in the two programs BELOW and ABOVE.

**Creating User-Defined Functions as Programs**

The defining procedure for a local variable structure can be either an algebraic or program object.

A program that consists solely of a local variable structure whose defining procedure is an algebraic expression is a user-defined function.

If a program begins with a local variable structure and has a program as the defining procedure, the complete program acts like a user-defined function in two ways: it takes numeric or symbolic arguments, and takes those arguments either from the stack or in algebraic syntax. However, it does not have a derivative. (The defining program must, like algebraic defining procedures, return only one result to the stack.)

There's an advantage to using a program as the defining procedure for a local variable structure: The program can contain commands not allowed in algebraic expressions. For example, loop structures are not allowed in algebraic expressions.
Using Tests and Conditional Structures

You can use commands and branching structures that let programs ask questions and make decisions. *Comparison functions* and *logical functions* test whether or not specified conditions exist. *Conditional structures* and *conditional commands* use test results to make decisions.

Testing Conditions

A test is an algebraic or a command sequence that returns a *test result* to the stack. A test result is either *true*—indicated by a value of 1—or it is *false*—indicated by a value of 0.

**To include a test in a program:**

- To use stack syntax, enter the two arguments, then enter the test command.
- To use algebraic syntax, enter the test expression (with ' delimiters).

You often use test results in conditional structures to determine which clause of the structure to execute. Conditional structures are described under “Using Conditional Structures and Commands” on page 1-20.

**Example:** Test whether or not $X$ is less than $Y$. To use stack syntax, enter $X \ Y \ <$. To use algebraic syntax, enter $'X<Y'$. (For both cases, if $X$ contains 5 and $Y$ contains 10, then the test is true and 1 is returned to the stack.)

Using Comparison Functions

Comparison functions compare two objects, using either stack syntax or algebraic syntax.
Comparison Functions

<table>
<thead>
<tr>
<th>Key</th>
<th>Programmable Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRG</td>
<td>T TEST (pages 1 and 2):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>==</td>
<td>Tests equality of two objects.</td>
</tr>
<tr>
<td></td>
<td>≠</td>
<td>Not equal.</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>Less than.</td>
</tr>
<tr>
<td></td>
<td>&gt;</td>
<td>Greater than.</td>
</tr>
<tr>
<td></td>
<td>≤</td>
<td>Less than or equal to.</td>
</tr>
<tr>
<td></td>
<td>≥</td>
<td>Greater than or equal to.</td>
</tr>
<tr>
<td></td>
<td>SAME</td>
<td>Identical. Like ==, but doesn't allow a comparison between the numerical value of an algebraic (or name) and a number. Also considers the wordsize of a binary integer.</td>
</tr>
</tbody>
</table>

The comparison commands return 1 (true) or 0 (false) based on the comparison—or an expression that can evaluate to 1 or 0. The order of the comparison is “level 2 test level 1,” where test is the comparison function.

All comparison commands except SAME return the following:

- If neither object is an algebraic or a name, returns 1 if the two objects are the same type and have the same value, or 0 otherwise. For example, if 6 is stored in X, \( \times 5 \leq \) puts 6 and 5 on the stack, then removes them and returns 0. (Lists and programs are considered to have the same value if the objects they contain are identical. For strings, “less than” means “alphabetically previous.”)

- If one object is an algebraic (or name) and the other object is an algebraic (or name) or a number, returns an expression that must be evaluated to get a test result based on numeric values. For example, if 6 is stored in X, 'X' 5 < returns 'X<5', then →NUM returns 0. (Note that == is used for comparisons, while = separates two sides of an equation.)

SAME returns 1 (true) if two objects are identical. For example, 'X+3' 4 SAME returns 0 regardless of the value of X because the algebraic 'X+3' is not identical to the real number 4. Binary integers
must have the same wordsize and the same value to be identical. For all object types other than algebraics, names, and binary integers, SAME works just like `==`.

You can use any comparison function (except SAME) in an algebraic by putting it between its two arguments. For example, if 6 is stored in X, `'X<5' `→ `NUM returns 0.

Using Logical Functions

Logical functions return a test result based on the outcomes of two previously executed tests. Note that these four functions interpret any nonzero argument as a true result.

<table>
<thead>
<tr>
<th>Logical Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keys</strong></td>
</tr>
<tr>
<td>PRG</td>
</tr>
<tr>
<td>AND</td>
</tr>
<tr>
<td>OR</td>
</tr>
<tr>
<td>XOR</td>
</tr>
<tr>
<td>NOT</td>
</tr>
</tbody>
</table>

AND, OR, and XOR combine two test results. For example, if 4 is stored in Y, Y > 5 AND returns 1. First, Y > 5 returns 1 to the stack. AND removes 1 and 5 from the stack, interpreting both as true results, and returns 1 to the stack.

NOT returns the logical inverse of a test result. For example, if 1 is stored in X and 2 is stored in Y, X < Y NOT returns 0.

You can use AND, OR, and XOR in algebraics as *prefix* functions. For example, `3<5 XOR 4>7` `→ `NUM returns 1.

You can use NOT as a *prefix* function in algebraics. For example, `NOT Z≤4` `→ `NUM returns 0 if Z = 2.
Testing Object Types

The TYPE command \((\text{PRG} \ 	ext{TEST} \ 	ext{NXT} \ 	ext{TYPE})\) takes any object as its argument and returns the number that identifies that object type. For example, "HELLO" TYPE returns 2, the value for a string object. See the table of object types in chapter 3, in the TYPE command, to find HP 48 objects and their corresponding type numbers.

Testing Linear Structure

The LININ command \((\text{PRG} \ 	ext{NXT} \ 	ext{TEST} \ 	ext{PREV} \ 	ext{LININ})\) takes an algebraic equation on level 2 and an variable on level 1 as arguments and returns 1 if the equation is linear for that variable, or 0 if it is not. For example, 'H+Y^2' LININ returns 1 because the equation is structurally linear for H. See the LININ command in chapter 3 for more information.

Using Conditional Structures and Commands

Conditional structures let a program make a decision based on the results of tests.

Conditional commands let you execute a true-clause or a false-clause (each of which are a single command or object).

These conditional structures and commands are contained in the PRG BRCH menu \((\text{PRG} \ 	ext{BRCH})\):

- IF ... THEN ... END structure.
- IF ... THEN ... ELSE ... END structure.
- CASE ... END structure.
- IFT (if-then) command.
- IFTE (if-then-else) function.

The IF ... THEN ... END Structure

The syntax for this structure is

\[
\langle ... \text{IF test-clause THEN true-clause END} ... \rangle
\]

IF ... THEN ... END executes the sequence of commands in the true-clause only if the test-clause evaluates to true. The test-clause can be a command sequence (for example, \(H \land B \leq \)) or an algebraic (for
example, \( \text{\texttt{\textasciitilde A \leq B}} \)). If the test-clause is an algebraic, it’s automatically evaluated to a number—you don’t need →NUM or EVAL.

IF begins the test-clause, which leaves a test result on the stack. THEN removes the test result from the stack. If the value is nonzero, the true-clause is executed—otherwise, program execution resumes following END. See “Conditional Examples” on page 1-23.

To enter IF ... THEN ... END in a program:

- Press \( \text{\texttt{PRG BRCH \leftarrow IF}} \).

The IFT Command

The IFT command takes two arguments: a test-result in level 2 and a true-clause object in level 1. If the test-result is true, the true-clause object is executed—otherwise, the two arguments are removed from the stack. See “Conditional Examples” on page 1-23.

To enter IFT in a program:

- Press \( \text{\texttt{PRG BRCH NEXT \leftarrow IFT}} \).

The IF ... THEN ... ELSE ... END Structure

The syntax for this structure is

\[
\text{\texttt{\textasciitilde \ldots IF test-clause}} \\
\text{\texttt{\textasciitilde THEN true-clause ELSE false-clause END \ldots \textasciitilde}}
\]

IF ... THEN ... ELSE ... END executes either the true-clause sequence of commands if the test-clause is true, or the false-clause sequence of commands if the test-clause is false. If the test-clause is an algebraic, it’s automatically evaluated to a number—you don’t need →NUM or EVAL.

IF begins the test-clause, which leaves a test result on the stack. THEN removes the test result from the stack. If the value is nonzero, the true-clause is executed—otherwise, the false-clause is executed. After the appropriate clause is executed, execution resumes following END. See “Conditional Examples” on page 1-23.

To enter IF ... THEN ... ELSE ... END in a program:

- Press \( \text{\texttt{PRG BRCH \rightarrow IF}} \).
The IFTE Function

The algebraic syntax for this function is

' IFTE( test, true-clause, false-clause ) '

If test evaluates true, the true-clause algebraic is evaluated—otherwise, the false-clause algebraic is evaluated.

You can also use the IFTE function with stack syntax. It takes three arguments: a test-result in level 3, a true-clause object in level 2, and a false-clause object in level 1. See “Conditional Examples” on page 1-23.

To enter IFTE in a program or in an algebraic:

- Press PRG BRCH NEXT ← IFTE.

The CASE . . . END Structure

The syntax for this structure is

« ... CASE
  test-clause\_1 \text{ THEN } true-clause\_1 \text{ END }
  test-clause\_2 \text{ THEN } true-clause\_2 \text{ END }
  \vdots
  test-clause\_n \text{ THEN } true-clause\_n \text{ END }
  default-clause \text{ (optional) }
END ... »

The CASE . . . END structure lets you execute a series of test-clause commands, then execute the appropriate true-clause sequence of commands. The first test that returns a true result causes execution of the corresponding true-clause, ending the CASE . . . END structure. Optionally, you can include after the last test a default-clause that’s executed if all the tests evaluate to false. If a test-clause is an algebraic, it’s automatically evaluated to a number—you don’t need →NUM or EVAL.

When CASE is executed, test-clause\_1 is evaluated. If the test is true, true-clause\_1 is executed, and execution skips to END. If test-clause\_1 is false, execution proceeds to test-clause\_2. Execution within the CASE structure continues until a true-clause is executed, or until all the test-clauses evaluate to false. If a default clause is included, it’s
executed if all the test-clauses evaluate to false. See "Conditional Examples" below.

To enter CASE ... END in a program:

1. Press \( \text{PRG} \) \( \text{BRCH} \) \( \leftarrow \) \( \text{CASE} \) to enter CASE ... THEN ... END ... END.
2. For each additional test-clause, move the cursor after a test-clause END and press \( \rightarrow \) \( \text{CASE} \) to enter THEN ... END.

Conditional Examples

These examples illustrate conditional structures in programs.

Example: One Conditional Action. The programs below test the value in level 1—if the value is positive, it’s made negative. The first program uses a command sequence as the test-clause:

\[
\text{DUP IF } 0 \text{ THEN NEG END } \]

The value on the stack must be duplicated because the > command removes two arguments from the stack (0 and the copy of the value made by DUP).

The following version uses an algebraic as the test clause:

\[
\langle \times \times \times \times \text{ IF } 'x>0' \text{ THEN NEG END } \rangle
\]

The following version uses the IFT command:

\[
\langle \text{DUP } 0 \langle \text{NEG } \rangle \text{ IFT } \rangle
\]

Example: One Conditional Action. This program multiplies two numbers if both are nonzero.
Program:

\[ \left( x \rightarrow x \rightarrow y \right) \]

Comments:

Creates local variables \( x \) and \( y \) containing the two numbers from the stack.

\[ \left( \begin{array}{c}
\text{IF} \\
\text{'}x\neq0' \\
\text{'}y\neq0' \\
\text{AND} \\
\text{THEN} \\
\times y * \\
\text{END} \\
\end{array} \right) \]

Starts the test-clause.
Tests one of the numbers and leaves a test result on the stack.
Tests the other number, leaving another test result on the stack.
Tests whether both tests were true.
Ends the test-clause, starts the true-clause.
Multiplies the two numbers together only if AND returns true.
Ends the true-clause.

The following program accomplishes the same task as the previous program:

\[ \left( x \rightarrow x \rightarrow y \left( \begin{array}{c}
\text{IF} \\
\text{'}x\AND y' \\
\text{THEN} \\
\times y * \\
\text{END} \\
\end{array} \right) \right) \]

The test-clause \( 'x\AND y' \) returns "true" if both numbers are nonzero.

The following version uses the IFT command:

\[ \left( x \rightarrow x \rightarrow y \left( \begin{array}{c}
\times y, 'x\AND y', 'x*y' \\
\text{IFT} \\
\end{array} \right) \right) \]
**Example: Two Conditional Actions.** This program takes a value \( x \) from the stack and calculates \( \frac{\sin x}{x} \). At \( x = 0 \) the division would error, so the program returns the limit value 1 in this case.

\[
\leftarrow x \leftarrow \text{IF} \ 'x\neq0' \ \text{THEN} \ x \ \text{SIN} \ x \ / \ \text{ELSE} \ 1 \ \text{END} \right]
\]

The following version uses IFTE algebraic syntax:

\[
\leftarrow x \ '\text{IFTE}(x\neq0,\sin(x)/x,1)' \right]
\]

**Example: Two Conditional Actions.** This program multiplies two numbers together if they’re both nonzero—otherwise, it returns the string "ZERO".

**Program:**

\[
\leftarrow \\
\rightarrow n1 \ n2 \\
\leftarrow \\
\text{IF} 'n1\neq0 \ \text{AND} \ n2\neq0' \\
\text{THEN} \ n1 \ n2 \ * \\
\text{ELSE} \ "ZERO" \\
\text{END} \\
\rightarrow \\
\rightarrow
\]

**Comments:**

- Creates the local variables.
- Starts the defining procedure.
- Starts the test clause.
- Tests \( n1 \) and \( n2 \).
- If both numbers are nonzero, multiplies the two values.
- Otherwise, returns the string "ZERO".
- Ends the conditional.
- Ends the defining procedure.
**Example: Two Conditional Actions.** This program tests if two numbers on the stack have the same value. If so, it drops one of the numbers and stores the other in variable V1—otherwise, it stores the number from level 1 in V1 and the number from level 2 in V2.

**Program:**
```
<
  IF
  DUP2
  SAME
  THEN
  DROP
  'V1' STO
  ELSE
  'V1' STO
  'V2' STO
  END
>
```

**Comments:**
- For the test clause, copies the numbers in levels 1 and 2 and tests if they have the same value.
- For the true clause, drops one of the numbers and stores the other in V1.
- For the false clause, stores the level 1 number in V1 and the level 2 number in V2.
- Ends the conditional structure.

Puts the program on the stack.
Stores it in TST.

Enter the numbers 26 and 52, then execute TST to compare their values. Because the two number aren’t equal, the VAR menu now contains two new variables V1 and V2.

```
26 ENTER 52
VAR TST
```

```
V2 | V1 | TST | TORSY | TORS | SPALV
```

1-26 Programming
**Example: Multiple Conditional Actions.** The following program stores the level 1 argument in a variable if the argument is a string, list, or program.

**Program:**

```
▼
  → y
  ▼

CASE
  y TYPE 2 SAME
  THEN y 'STR' STO END
  y TYPE 5 SAME
  THEN y 'LIST' STO END
  y TYPE 8 SAME
  THEN y 'PROG' STO END

END
▼
▼
```

**Comments:**

- Defines local variable *y*.
- Starts the defining procedure.
- Starts the case structure.
- Case 1: If the argument is a string, stores it in *STR*.
- Case 2: If the argument is a list, stores it in *LIST*.
- Case 3: If the argument is a program, stores it in *PROG*.
- Ends the case structure.
- Ends the defining procedure.

---

**Using Loop Structures**

You can use loop structures to execute a part of a program repeatedly. To specify in advance how many times to repeat the loop, use a *definite loop*. To use a test to determine whether or not to repeat the loop, use an *indefinite loop*.

*Loop structures* let a program execute a sequence of commands several times. Loop structures are built with commands—called structure words—that work only when used in proper combination with each other. These loop structure commands are contained in the PRG BRCH menu ((PRG) BRCH):

- **START ... NEXT** and **START ... STEP**.
- **FOR ... NEXT** and **FOR ... STEP**.
- **DO ... UNTIL ... END**.
- **WHILE ... REPEAT ... END**.
In addition, the $\Sigma$ function provides an alternative to definite loop structures for summations.

**Using Definite Loop Structures**

Each of the two definite loop structures has two variations:

- **NEXT.** The counter increases by 1 for each loop.
- **STEP.** The counter increases or decreases by a specified amount for each loop.

**The START ... NEXT Structure**

The syntax for this structure is

```
... start finish: START loop-clause NEXT ...
```

START ... NEXT executes the loop-clause sequence of commands one time for each number in the range start to finish. The loop-clause is always executed at least once.

![Syntax Flowchart](image-url)
START takes two numbers (start and finish) from the stack and stores them as the starting and ending values for a loop counter. Then, the loop-clause is executed. NEXT increments the counter by 1 and tests to see if its value is less than or equal to finish. If so, the loop-clause is executed again—otherwise, execution resumes following NEXT.

To enter START ... NEXT in a program:

- Press PRG BRCH START.

Example: The following program creates a list containing 10 copies of the string "ABC":

```
« 1 10 START "ABC" NEXT 10 +LIST »
```
The START ... STEP Structure

The syntax for this structure is

« ... start finish START loop-clause increment STEP ... »

START ... STEP executes the loop-clause sequence just like START ... NEXT does—except that the program specifies the increment value for the counter, rather than incrementing by 1. The loop-clause is always executed at least once.

START ... STEP Structure

START takes two numbers (start and finish) from the stack and stores them as the starting and ending values of the loop counter. Then the loop-clause is executed. STEP takes the increment value from the stack and increments the counter by that value. If the argument
of STEP is an algebraic or a name, it's automatically evaluated to a number.

The increment value can be positive or negative. If it's positive, the loop is executed again if the counter is less than or equal to finish. If the increment value is negative, the loop is executed if the counter is greater than or equal to finish. Otherwise, execution resumes following STEP. In the previous flowchart, the increment value is positive.

To enter START ... STEP in a program:

- Press \texttt{(PRG) BRCH \rightarrow START}.

Example: The following program takes a number \( z \) from the stack and calculates the square of that number several times (\( z/3 \) times):

\[
\texttt{\{ DUP \ \times \ \times \ \times \ 1 \ \texttt{START} \ \times \ \texttt{SQ} \ -3 \ \texttt{STEP} \ \}}
\]
The FOR ... NEXT Structure

The syntax for this structure is

« ... start finish FOR counter loop-clause NEXT ... »

FOR ... NEXT executes the loop-clause program segment one time for each number in the range start to finish, using local variable counter as the loop counter. You can use this variable in the loop-clause. The loop-clause is always executed at least once.

FOR takes start and finish from the stack as the beginning and ending values for the loop counter, then creates the local variable counter as a loop counter. Then the loop-clause is executed—counter can appear within the loop-clause. NEXT increments counter-name by one, and then tests whether its value is less than or equal to finish. If so, the loop-clause is repeated (with the new value of counter)—otherwise,
execution resumes following NEXT. When the loop is exited, counter is purged.

To enter FOR ... NEXT in a program:

Press PRG BRCH FOR.

Example: The following program places the squares of the integers 1 through 5 on the stack:

```plaintext
« 1 5 FOR j j SQ NEXT »
```

Example: The following program takes the value z from the stack and computes the integer powers i of z. For example, when z = 12 and start and finish are 3 and 5 respectively, the program returns 12³, 12⁴, and 12⁵. It requires as inputs start and finish in levels 3 and 2, and z in level 1. (÷ × removes z from the stack, leaving start and finish there as arguments for FOR.)

```plaintext
« ÷ × « FOR n 'n^n' EVAL NEXT » »
```
The FOR ... STEP Structure

The syntax for this structure is

```
... start finish FOR counter loop-clause increment STEP ... *
```

FOR ... STEP executes the loop-clause sequence just like FOR ... NEXT does—except that the program specifies the increment value for counter, rather than incrementing by 1. The loop-clause is always executed at least once.

---

FOR ... STEP Structure

FOR takes start and finish from the stack as the beginning and ending values for the loop counter, then creates the local variable counter as a loop counter. Next, the loop-clause is executed—counter can appear within the loop-clause. STEP takes the increment value from the
stack and increments counter by that value. If the argument of STEP is an algebraic or a name, it's automatically evaluated to a number.

The increment value can be positive or negative. If the increment is positive, the loop is executed again if counter is less than or equal to finish. If the increment is negative, the loop is executed if counter is greater than or equal to finish. Otherwise, counter is purged and execution resumes following STEP. In the previous flowchart, the increment value is positive.

To enter FOR ... STEP in a program:

- Press (PRG) BRCH (FOR).

Example: The following program places the squares of the integers 1, 3, 5, 7, and 9 on the stack:

```
< 1 9 FOR \times \times SQ 2 STEP >
```

Example: The following program takes n from the stack, and returns the series of numbers 1, 2, 4, 8, 16, ..., n. If n isn't in the series, the program stops at the last value less than n.

```
< 1 \text{SWAP} FOR n n n STEP >
```

The first n is the local variable declaration for the FOR loop. The second n is put on the stack each iteration of the loop. The third n is used by STEP as the step increment.
Using Indefinite Loop Structures

The DO ... UNTIL ... END Structure

The syntax for this structure is

```
  ... DO loop-clause UNTIL test-clause END ...
```

DO ... UNTIL ... END executes the loop-clause sequence repeatedly until test-clause returns a true (nonzero) result. Because the test-clause is executed after the loop-clause, the loop-clause is always executed at least once.

**Syntax Flowchart**

**DO ... UNTIL ... END Structure**

DO starts execution of the loop-clause. UNTIL marks the end of the loop-clause. The test-clause leaves a test result on the stack. END removes the test result from the stack. If its value is zero, the loop-clause is executed again—otherwise, execution resumes following END. If the argument of END is an algebraic or a name, it's automatically evaluated to a number.
To enter DO ... UNTIL ... END in a program:

Press PRG BRCH ⇧ DO...

Example: The following program calculates \( n + 2n + 3n + \ldots \) for a value of \( n \). The program stops when the sum exceeds 1000, and returns the sum and the coefficient of \( n \).

Program:

\[
<
\text{DUP } 1
\rightarrow n \leftarrow c
<
\text{DO}
'c' \text{ INCR}
\]
\[
\text{n } \times 's' \text{ STO+}
\]
UNTIL
\[
\leq 1000 >
\text{END}
\leq c
\]
>  

Comments:

Duplicates \( n \), stores the value into \( n \) and \( s \), and initializes \( c \) to 1.

Starts the defining procedure.

Starts the loop-clause.

Increments the counter by 1. (See "Using Loop Counters" on page 1-39.)

Calculates \( c \times n \) and adds the product to \( s \).

Starts the test clause.

Repeats loop until \( s > 1000 \).

Ends the test-clause.

Puts \( s \) and \( c \) on the stack.

Ends the defining procedure.
The WHILE ... REPEAT ... END Structure

The syntax for this structure is

« ... WHILE test-clause REPEAT loop-clause END ... »

WHILE ... REPEAT ... END repeatedly evaluates test-clause and executes the loop-clause sequence if the test is true. Because the test-clause is executed before the loop-clause, the loop-clause is not executed if the test is initially false.

WHILE starts execution of the test-clause, which returns a test result to the stack. REPEAT takes the value from the stack. If the value is nonzero, execution continues with the loop-clause—otherwise, execution resumes following END. If the argument of REPEAT is an algebraic or a name, it's automatically evaluated to a number.

To enter WHILE ... REPEAT ... END in a program:

• Press [PRG] [BRCH] \(\leftrightarrow\) WHILE.
Example: The following program starts with a number on the stack, and repeatedly performs a division by 2 as long as the result is evenly divisible. For example, starting with the number 24, the program computes 12, then 6, then 3.

```
WHILE DUP 2 MOD 0 == REPEAT 2 / DUP END DROP
```

Example: The following program takes any number of vectors or arrays from the stack and adds them to the statistics matrix. (The vectors and arrays must have the same number of columns.) WHILE ... REPEAT ... END is used instead of DO ... UNTIL ... END because the test must be done before the addition. (If only vectors or arrays with the same number of columns are on the stack, the program errors after the last vector or array is added to the statistics matrix.)

```
WHILE DUP TYPE 3 == REPEAT 2+ END
```

Using Loop Counters

For certain problems you may need a counter inside a loop structure to keep track of the number of loops. (This counter isn’t related to the counter variable in a FOR ... NEXT/STEP structure.) You can use any global or local variable as a counter. You can use the INCR or DECR command to increment or decrement the counter value and put its new value on the stack.

The syntax for INCR and DECR is

```
... \texttt{ 'variable' INCR ... }
```

or

```
... \texttt{ 'variable' DECR ... }
```

To enter INCR or DECR in a program:

- Press \texttt{(MEMORY) ARITH INCR or DECR}.

The INCR and DECR commands take a global or local variable name (with \texttt{'} delimiters) as their argument—the variable must contain a real number. The command does the following:

1. Changes the value stored in the variable by $+1$ or $-1$.
2. Returns the new value to the stack.
Examples: If \( c \) contains the value 5, then \( 'c' \) \texttt{INCR} stores 6 in \( c \) and returns 6 to the stack.

The following program takes a maximum of five vectors from the stack and adds them to the current statistics matrix.

**Program:**

\[
\begin{align*}
\times \\
0 \rightarrow c \\
\llcorner \\
\text{WHILE} \\
\text{DUP TYPE 3 ==} \\
'c' \text{ INCR} \\
5 \leq \text{AND} \\
\text{REPEAT} \\
\Sigma+ \\
\text{END} \\
\gg \\
\gg
\end{align*}
\]

**Comments:**

Stores 0 in local variable \( c \).
Starts the defining procedure.
Starts the test clause.
Returns true if level 1 contains a vector.
Increments and returns the value in \( c \).
Returns true if the counter \( c \leq 5 \).
Returns true if the two previous test results are true.

Adds the vector to \( \Sigma DAT \).
Ends the structure.
Ends the defining procedure.

**Using Summations Instead of Loops**

For certain calculations that involve summations, you can use the \( \Sigma \) function instead of loops. You can use \( \Sigma \) with stack syntax or with algebraic syntax. \( \Sigma \) automatically repeats the addition for the specified range of the index variable—without using a loop structure.

**Example:** The following programs take an integer upper limit \( n \) from the stack, then find the summation

\[
\sum_{j=1}^{n} j^2
\]

One program uses a \texttt{FOR . . . NEXT} loop—the other uses \( \Sigma \).
Program:
«
0 1 ROT
FOR j
j SQ +
NEXT
»

Comments:
Initializes the summation and puts the limits in place.
Loops through the calculation.

Program:
«
→ n
'Σ(j=1,n,j^2)'
»

Comments:
Uses Σ to calculate the summation.

Example: The following program uses ΣLIST to calculate the summation of all elements of a vector or matrix. The program takes from the stack an array or a name that evaluates to an array, and returns the summation.

Program:
«
OBJ→
1
+
πLIST
→LIST ΣLIST
»

Comments:
Finds the dimensions of the array and leaves it in a list on level 1.
Adds 1 to the list. (If the array is a vector, the list on level 1 has only one element. ΠLIST will error if the list has fewer than two elements.)
Multiplies all of the list elements together.
Converts the array elements into a list, and sums them.
Using Flags

You can use flags to control calculator behavior and program execution. You can think of a flag as a switch that is either on (set) or off (clear). You can test a flag's state within a conditional or loop structure to make a decision. Because certain flags have unique meanings for the calculator, flag tests expand a program's decision-making capabilities beyond that available with comparison and logical functions.

Types of Flags

The HP 48 has two types of flags:

- **System flags.** Flags —1 through —64. These flags have predefined meanings for the calculator.
- **User flags.** Flags 1 through 64. User flags are not used by any built-in operations. What they mean depends entirely on how the program uses them.

Appendix C lists the 64 system flags and their definitions. For example, system flag —40 controls the clock display—when this flag is clear (the default state), the clock is not displayed—when this flag is set, the clock is displayed. (When you press `CLK` in the `Modes` menu, you are setting or clearing flag —40.)

When you set user flag 1 through 5, the corresponding annunciator is turned on. Certain plug-in cards may use user-flags in the range 31 through 64.

Setting, Clearing, and Testing Flags

Flag commands take a flag number from the stack—an integer 1 through 64 (for user flags) or —1 through —64 (for system flags).

**To set, clear, or test a flag:**

1. Enter the flag number (positive or negative).
2. Execute the flag command—see the table below.
Flag Commands

<table>
<thead>
<tr>
<th>Key</th>
<th>Programmable Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRG</td>
<td>TEST (NXT) (NXT) or (M O D E S) FLAG :</td>
<td>Sets the flag.</td>
</tr>
<tr>
<td>CF</td>
<td>SF</td>
<td>Clears the flag.</td>
</tr>
<tr>
<td>FS?</td>
<td>CF</td>
<td>Returns 1 (true) if the flag is set, or 0 (false) if the flag is clear.</td>
</tr>
<tr>
<td>FC?</td>
<td>FS?</td>
<td>Returns 1 (true) if the flag is clear, or 0 (false) if the flag is set.</td>
</tr>
<tr>
<td>FS?C</td>
<td>FC?</td>
<td>Tests the flag (returns true if the flag is set), then clears the flag.</td>
</tr>
<tr>
<td>FC?C</td>
<td>FC?C</td>
<td>Tests the flag (returns true if the flag is clear), then clears the flag.</td>
</tr>
</tbody>
</table>

**Example: System Flag.** The following program sets an alarm for June 6, 1993 at 5:05 PM. It first tests the status of system flag \(-42\) (Date Format flag) in a conditional structure and then supplies the alarm date in the current date format, based on the test result.

**Program:**

```
* IF
 -42 FC? THEN 6.151993 ELSE 15.061993 END
17.05 "TEST COMPLETE" 3 +LIST STOALARM
```

**Comments:**

Tests the status of flag \(-42\), the Date Format flag.
If flag \(-42\) is clear, supplies the date in month/day/year format.
If flag \(-42\) is set, supplies the date in day.month.year format.
Sets the alarm: 17.05 is the alarm time and “TEST COMPLETE” is the alarm message.

**Example: User Flag.** The following program returns either the fractional or integer part of the number in level 1, depending on the state of user flag 10.
Program:  
<  
 IF  
   10 FS?  
 THEN  
   IP  
 ELSE  
   FP  
 END  
>

Comments:  
Starts the conditional.  
Tests the status of user flag 10.  
If flag 10 is set, returns the integer part.  
If flag 10 is clear, returns the fractional part.  
Ends the conditional.

To use this program, you enter a number, either set flag 10 (to get the integer part) or clear flag 10 (to get the fractional part), then run the program.

Recalling and Storing the Flag States

If you have a program that changes the state of a flag during execution, you may want it to save and restore original flag states.

The RCLF (recall flags) and STOF (store flags) commands let you recall and store the states of the HP 48 flags. For these commands, a 64-bit binary integer represents the states of 64 flags—each 0 bit corresponds to a flag that’s clear, each 1 bit corresponds to a flag that’s set. The rightmost (least significant) bit corresponds to system flag −1 or user flag 1.

To recall the current flag states:

1. Execute RCLF ([MODES] [FLAG] [NXT] RCLF).

RCLF returns a list containing two 64-bit binary integers representing the current states of the system and user flags:

{ #n_s #n_u }

To change the current flag states:

1. Enter the flag-state argument—see below.
2. Execute STOF ([MODES] [FLAG] [NXT] STOF).
STOF sets the current states of flags based on the flag-state argument:

\#ns \quad \text{Changes the states of only the system flags.}
\langle \#ns \#nu \rangle \quad \text{Changes the states of the system and user flags.}

Example: The program \textit{PRE\textsc{serve}} on page 2-8 uses RCLF and STOF.

---

**Using Subroutines**

Because a program is itself an object, it can be used in another program as a subroutine. When program \( B \) is used by program \( A \), program \( A \) \textit{calls} program \( B \), and program \( B \) is a \textit{subroutine} in program \( A \).

Example: The program \textit{TORS\textsc{A}} calculates the surface area of a torus of inner radius \( a \) and outer radius \( b \). \textit{TORS\textsc{A}} is used as a subroutine in a second program \textit{TORS\textsc{V}}, which calculates the volume of a torus.

![Diagram of a torus](https://via.placeholder.com/150)

The surface area and volume are calculated by

\[ A = \pi^2(b^2 - a^2) \quad V = \frac{1}{4} \pi^2(b^2 - a^2)(b - a) \]

(The quantity \( \pi^2(b^2 - a^2) \) in the second equation is the surface area of a torus calculated by \textit{TORS\textsc{A}.})
Here are the stack diagram and program listing for \textit{TORSA}.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>→</td>
<td>surface area</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Program:**
\[
\langle \\
\rightarrow a \ b \\
\pi^2*(b^2-a^2) \\
\rightarrow \text{NUM} \\
\times \\
\text{ENTER} \\
\hat{\text{TORSV}} \ \text{STO}
\]

**Comments:**
- Creates local variables $a$ and $b$.
- Calculates the surface area.
- Converts algebraic to a number.
- Puts the program on the stack.
- Stores the program in \textit{TORSA}.

Here is a stack diagram and program listing for \textit{TORSV}.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>→</td>
<td>volume</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Program:**
\[
\langle \\
\rightarrow a \ b \\
\times \\
\hat{a} \ b \ \text{TORSA} \\
\b \ a - * \ 4/ \\
\rightarrow \\
\rightarrow \\
\text{ENTER} \\
\hat{\text{TORSV}} \ \text{STO}
\]

**Comments:**
- Creates local variables $a$ and $b$.
- Starts a program as the defining procedure.
- Puts the numbers stored in $a$ and $b$ on the stack, then calls \textit{TORSA} with those arguments.
- Completes the volume calculation using the surface area.
- Ends the defining procedure.
- Puts the program on the stack.
- Stores the program in \textit{TORSV}.

1-46 Programming
Now use \textit{TORSV} to calculate the volume of a torus of inner radius \(a = 6\) and outer radius \(b = 8\).

\begin{verbatim}
6 \text{ ENTER} 8
\text{ VAR TORSV}
\end{verbatim}

\begin{verbatim}
1: 138.1744616
\end{verbatim}

---

\section*{Single-Stepping through a Program}

It's easier to understand how a program works if you execute it step by step, observing the effect of each step. Doing this can help you debug your own programs or understand programs written by others.

\textbf{To single-step from the start of a program:}

1. Put the program or program name in level 1 (or the command line).
2. Press \texttt{PRG(NXT RUN DEBUG} to start and immediately suspend execution.
   \texttt{HALT} appears in the status area.
3. Take any action:
   \begin{itemize}
   \item To see the next program step displayed in the status area and then executed, press \texttt{SST}.
   \item To display but not execute the next one or two program steps, press \texttt{NEXT}.
   \item To continue with normal execution, press \texttt{CONT}.
   \item To abandon further execution, press \texttt{KILL}.
   \end{itemize}
4. Repeat the previous step as desired.

\textbf{To turn off the HALT annunciator at any time:}

- Press \texttt{PRG(NXT RUN KILL}.

\textbf{Example:} Execute program \textit{TORSV} step by step. Use \(a = 6\) and \(b = 8\).
Select the VAR menu and enter the data. Enter the program name and start the debugging. HALT indicates program execution is suspended.

Display and execute the first program step. Notice that it takes the two arguments from the stack and stored them in local variables \( a \) and \( b \).

Continue single-stepping until the status area shows the current directory. Watch the stack and status area as you single-step through the program.

To single-step from the middle of a program:

1. Insert a HALT command in the program where you want to begin single-stepping.
2. Execute the program normally. The program stops when the HALT command is executed, and the HALT annunciator appears.
3. Take any action:
   - To see the next program step displayed in the status area and then executed, press \( \text{ SST} \).
   - To display but not execute the next one or two program steps, press \( \text{ NEXT} \).
   - To continue with normal execution, press \( \text{ CONT} \).
   - To abandon further execution, press \( \text{ KILL} \).
4. Repeat the previous step as desired.
When you want the program to run normally again, remove the HALT command from the program.

**To single-step when the next step is a subroutine:**

- To execute the subroutine in one step, press \( \text{SST} \).
- To execute the subroutine step-by-step, press \( \text{SST}^+ \).

\( \text{SST} \) executes the next step in a program—if the next step is a subroutine, \( \text{SST} \) executes the subroutine in one step. \( \text{SST}^+ \) works just like \( \text{SST} \)—except if the next program step is a subroutine, it single-steps to the first step in the subroutine.

**Example:** In the previous example, you used \( \text{SST} \) to execute subroutine \( \text{TORSA} \) in one step. Now execute program \( \text{TORSV} \) step by step to calculate the volume of a torus of radii \( a = 10 \) and \( b = 12 \). When you reach subroutine \( \text{TORSA} \), execute it step by step.

Select the VAR menu and enter the data. Enter the program name and start the debugging. Execute the first four steps of the program, then check the next step.

```
\(<\) CLEAR \( \) VAR
10 \( \) ENTER \( \) 12
\( \) TORSV
PRG \( \) NXT \( \) RUN \( \) DEBUG
\( \) SST+ (4 times)
\( \) NEXT
```

The next step is \( \text{TORSA} \). Single-step into \( \text{TORSA} \), then check that you’re at the first step of \( \text{TORSA} \).

```
\( \) SST+ \( \) NEXT
```

Press \(<\) CONT \( \) \(<\) CONT \) to complete subroutine and program execution.

The following table summarizes the operations for single-stepping through a program.
## Single-Step Operations

<table>
<thead>
<tr>
<th>Key</th>
<th>Programmable Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[PRG]</td>
<td>[NXT] RUN:</td>
<td>Starts program execution, then suspends it as if HALT were the first program command. Takes as its argument the program or program name in level 1.</td>
</tr>
<tr>
<td>[ ] [ ]</td>
<td>[ ] [ ]</td>
<td>Executes the next object or command in the suspended program.</td>
</tr>
<tr>
<td>[ ] [ ]</td>
<td>[ ] [ ]</td>
<td>Same as [ ] [ ], except if the next program step is a subroutine, single-steps to the first step in that subroutine.</td>
</tr>
<tr>
<td>[ ]</td>
<td>[ ] NEXT</td>
<td>Displays the next one or two objects, but does not execute them. The display persists until the next keystroke.</td>
</tr>
<tr>
<td>[ ] [ ]</td>
<td>[ ] [ ]</td>
<td>HALT Suspends program execution at the location of the HALT command in the program.</td>
</tr>
<tr>
<td>[ ] [ ]</td>
<td>[ ] [ ]</td>
<td>KILL Cancels all suspended programs and turns off the HALT annunciator.</td>
</tr>
<tr>
<td>[ ] [ ]</td>
<td>[ ] [ ]</td>
<td>CONT Resumes execution of a halted program.</td>
</tr>
</tbody>
</table>

## Trapping Errors

If you attempt an invalid operation from the keyboard, the operation is not executed and an error message appears. For example, if you execute + with a vector and a real number on the stack, the HP 48 returns the message + Error: Bad Argument Type and returns the arguments to the stack (if Last Arguments is enabled).

In a program, the same thing happens, but program execution is also aborted. If you anticipate error conditions, your program can process them without interrupting execution.
For simple programs, you can run the program again if it stops with an error. For other programs, you can design them to *trap* errors and continue executing. You can also create user-defined errors to trap certain conditions in programs. The error trapping commands are located in the PRG ERROR menu.

**Causing and Analyzing Errors**

Many conditions are automatically recognized by the HP 48 as error conditions—and they’re automatically treated as errors in programs.

You can also define conditions that cause errors. You can cause a *user-defined error* (with a user-defined error message)—or you can cause a built-in error. Normally, you’ll include a conditional or loop structure with a test for the error condition—and if it occurs, you’ll cause the user-defined or built-in error to occur.

**To cause a user-defined error to occur in a program:**

1. Enter a string (with " " delimiters) containing the desired error message.
2. Enter the DOERR command (PRG ERROR menu).

**To artificially cause a built-in error to occur in a program:**

1. Enter the error number (as a binary integer or real number) for the error.
2. Enter the DOERR command (PRG ERROR menu).

If DOERR is trapped in an IFERR structure (described in the next topic), execution continues. If it’s not trapped, execution is abandoned at the DOERR command and the error message appears.

**To analyze an error in a program:**

- To get the error number for the last error, execute ERRN (PRG ERROR menu).
- To get the error message for the last error, execute ERRM (PRG ERROR menu).
- To clear the last-error information, execute ERR0 (PRG ERROR menu).

The error number for a user-defined error is `#70000h`. See the list of built-in error numbers in appendix A, “Error and Status Messages.”
**Example:** The following program aborts execution if the list in level 1 contains three objects.

```
«
OBJ
IF 3 SAME
THEN "3 OBJECTS IN LIST" DOERR
END
»
```

The following table summarizes error trapping commands.

<table>
<thead>
<tr>
<th>Key</th>
<th>Programmable Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRG (NXT)</strong> <strong>ERROR:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DOERR</strong></td>
<td><strong>DOERR</strong></td>
<td>Causes an error. For a string in level 1, causes a user-defined error: the calculator behaves just as if an ordinary error has occurred. For a binary integer or real number in level 1, causes the corresponding built-in error. If the error isn’t trapped in an IFERR structure, DOERR displays the message and abandons program execution. (For 0 in level 1, abandons execution without updating the error number or message—like [CANCEL].)</td>
</tr>
<tr>
<td><strong>ERRN</strong></td>
<td><strong>ERRN</strong></td>
<td>Returns the error number, as a binary integer, of the most recent error. Returns #0 if the error number was cleared by ERR0.</td>
</tr>
<tr>
<td><strong>ERRM</strong></td>
<td><strong>ERRM</strong></td>
<td>Returns the error message (a string) for the most recent error. Returns an empty string if the error number was cleared by ERR0.</td>
</tr>
<tr>
<td><strong>ERR0</strong></td>
<td><strong>ERR0</strong></td>
<td>Clears the last error number and message.</td>
</tr>
</tbody>
</table>
Making an Error Trap

You can construct an error trap with one of the following conditional structures:

- IFERR ... THEN ... END.
- IFERR ... THEN ... ELSE ... END.

The IFERR ... THEN ... END Structure

The syntax for this structure is

```
< ... IFERR trap-clause THEN error-clause END ... »
```

The commands in the error-clause are executed only if an error is generated during execution of the trap-clause. If an error occurs in the trap-clause, the error is ignored, the remainder of the trap-clause is skipped, and program execution jumps to the error-clause. If no errors occur in the trap-clause, the error-clause is skipped and execution resumes after the END command.

To enter IFERR ... THEN ... END in a program:

- Press \[PRG\] \[NXT\] \[ERROR\] \[IFERR\].

**Example:** The following program takes any number of vectors or arrays from the stack and adds them to the statistics matrix. However, the program stops with an error if a vector or array with a different number of columns is encountered. In addition, if only vectors or arrays with the same number of columns are on the stack, the program stops with an error after the last vector or array has been removed from the stack.

```
< WHILE DUP TYPE 3 == REPEAT ∑+ END »
```

In the following revised version, the program simply attempts to add the level 1 object to the statistics matrix until an error occurs. Then, it ends by displaying the message \[DONE\].
Program: Comments:

```
<
  IFERR
  WHILE 1
  REPEAT 2+
  END
  THEN "DONE" 1 DISP 1 FREEZE
  END
>
```

The IFERR ... THEN ... ELSE ... END Structure

The syntax for this structure is

```
< ... IFERR trap-clause
   THEN error-clause ELSE normal-clause END ... >
```

The commands in the error-clause are executed only if an error is generated during execution of the trap-clause. If an error occurs in the trap-clause, the error is ignored, the remainder of the trap-clause is skipped, and program execution jumps to the error-clause. If no errors occur in the trap-clause, execution jumps to the normal-clause at the completion of the trap-clause.

To enter IFERR ... THEN ... ELSE ... END in a program:

- Press PRGNXT ERROR IFERR.

Example: The following program prompts for two numbers, then adds them. If only one number is supplied, the program displays an error message and prompts again.
Program:

```
DO
    "KEY IN a AND b" " "
    INPUT OBJ>
UNTIL
    IFERR
    +
    THEN
        ERRM 5 DISP
        2 WAIT
        0
    ELSE
        1
    END
END
```

Comments:

Begins the main loop.
Prompts for two numbers.

Starts the loop test clause.
The error trap contains only the + command.
If an error occurs, recalls and displays the Too Few Arguments message for 2 seconds, then puts 0 (false) on the stack for the main loop.
If no error occurs, puts 1 (true) on the stack for the main loop.
Ends the error trap.
Ends the main loop. If the error trap left 0 (false) on the stack, the main loop repeats—otherwise, the program ends.

---

Input

A program can stop for user input, then resume execution, or can use choose boxes or input forms (dialog boxes) for input. You can use several commands to get input:

- PROMPT (CONT) to resume).
- DISP FREEZE HALT (CONT) to resume).
- INPUT (ENTER to resume).
- INFORM
- CHOOSE
Data Input Commands

<table>
<thead>
<tr>
<th>Key</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORM</td>
<td>INFORM</td>
<td>Creates a user-defined input form.</td>
</tr>
<tr>
<td>NOVAL</td>
<td>NOVAL</td>
<td>Place holder for the INFORM command. Returned when a value is not present in an input form field.</td>
</tr>
<tr>
<td>CHOOSE</td>
<td>CHOOSE</td>
<td>Creates a user-defined choose box.</td>
</tr>
<tr>
<td>KEY</td>
<td>KEY</td>
<td>Returns a test result to level 1 and, if a key is pressed, the location of that key (level 2).</td>
</tr>
<tr>
<td>WAIT</td>
<td>WAIT</td>
<td>Suspends program execution for a specified duration (in seconds, level 1).</td>
</tr>
<tr>
<td>INPUT</td>
<td>INPUT</td>
<td>Suspends program execution for data input.</td>
</tr>
<tr>
<td>PROMPT</td>
<td>PROMPT</td>
<td>Halts program execution for data input.</td>
</tr>
</tbody>
</table>

Using PROMPT ... CONT for Input

PROMPT uses the status area for prompting, and allows the user to use normal keyboard operations during input.

To enter PROMPT in a program:

1. Enter a string (with " " delimiters) to be displayed as a prompt in the status area.
2. Enter the PROMPT command (PRG IN menu).

   « ... "prompt-string" PROMPT ... »

PROMPT takes a string argument from level 1, displays the string (without the " " delimiters) in the status area, and halts program execution. Calculator control is returned to the keyboard.

When execution resumes, the input is left on the stack as entered.

To respond to PROMPT while running a program:

1. Enter your input—you can use keyboard operations to calculate the input.
2. Press \( \text{CONT} \).
The message remains until you press \( \text{ENTER} \) or \( \text{CANCEL} \) or until you update the status area.

**Example:** If you execute this program segment

\[
\text{"ABC?" PROMPT }
\]

the display looks like this:

Example: The following program, \( TPROMPT \), prompts you for the dimensions of a torus, then calls program \( TORSA \) (from page 1-45) to calculate its surface area. You don’t have to enter data on the stack prior to program execution.

**Program:**

\[
\text{"ENTER } a, b \text{ IN ORDER:"
}
\]

**Comments:**

Puts the prompting string on the stack.

Displays the string in the status area, halts program execution, and returns calculator control to the keyboard.

Executes \( TORSA \) using the just-entered stack arguments.

Stores the program in \( TPROMPT \).

Execute \( TPROMPT \) to calculate the volume of a torus with inner radius \( a = 8 \) and outer radius \( b = 10 \).
Execute `TPROMPT`. The program prompts you for data.

```
(\rightarrow) CLEAR
VAR TPROM
```

Enter the inner and outer radii. After you press `ENTER`, the prompt message is cleared from the status area.

```
8 ENTER 10
```

Continue the program.

```
(\leftarrow) CONT
```

Note that when program execution is suspended by `PROMPT`, you can execute calculator operations just as you did before you started the program. If the outer radius $b$ of the torus in the previous example is measured as 0.83 feet, you can convert that value to inches while the program is suspended for data input by pressing `.83 ENTER 12 \times`, then `\leftarrow CONT`.

### Using DISP FREEZE HALT . . . CONT for Input

DISP FREEZE HALT lets you control the entire display during input, and allows the user to use normal keyboard operations during input.

**To enter DISP FREEZE HALT in a program:**

1. Enter a string or other object to be displayed as a prompt.
2. Enter a number specifying the line to display it on.
3. Enter the `DISP` command (PRG OUT menu).
4. Enter a number specifying the areas of the display to “freeze.”
5. Enter the `FREEZE` command (PRG OUT menu).
6. Enter the `HALT` command (PRG OUT menu).
DISP displays an object in a specified line of the display. DISP takes two arguments from the stack: an object from level 2, and a display-line number 1 through 7 from level 1. If the object is a string, it's displayed without the **delimiters. The display created by DISP persists only as long as the program continues execution—if the program ends or is suspended by HALT, the calculator returns to the normal stack environment and updates the display. However, you can use FREEZE to retain the prompt display.

FREEZE "freezes" display areas so they aren't updated until a *key press*. Argument n in level 1 is the sum of the codes for the areas to be frozen: 1 for the status area, 2 for the stack/command line area, 4 for the menu area.

HALT suspends program execution at the location of the HALT command and turns on the HALT annunciator. Calculator control is returned to the keyboard for normal operations.

When execution resumes, the input remains on the stack as entered.

**To respond to HALT while running a program:**

1. Enter your input—you can use keyboard operations to calculate the input.
2. Press `[CONT].

**Example:** If you execute this program segment

```
"ABC=DEF=GHI" CLLCD 1 DISP 3 FREEZE HALT
```

the display looks like this:

```
ABC
DEF
GHI
```

(The ® in the previous program is the calculator's representation for the " newline character after you enter a program on the stack.)
Using INPUT . . . ENTER for Input

INPUT lets you use the stack area for prompting, lets you supply default input, and prevents the user from using normal stack operations or altering data on the stack.

To enter INPUT in a program:

1. Enter a string (with " " delimiters) to be displayed as a prompt at the top of the stack area.
2. Enter a string or list (with delimiters) that specifies the command-line content and behavior—see below.
3. Enter the INPUT command (PRG IN menu).
4. Enter OBJ→ (PRG TYPE menu) or other command that processes the input as a string object.

```
"prompt-string" "command-line" INPUT OBJ→ ... »
```

or

```
"prompt-string" {command-line} INPUT OBJ→ ... »
```

INPUT, in its simplest form, takes two strings as arguments—see the list of additional options following. INPUT blanks the stack area, displays the contents of the level-2 string at the top of the stack area, and displays the contents of the level-1 string in the command line. It then activates Program-entry mode, puts the insert cursor after the string in the command line, and suspends execution.

When execution resumes, the input is returned to level 1 as a string object, called the result string.

To respond to INPUT while running a program:

1. Enter your input. (You can’t execute commands—they’re simply echoed in the command line.)
2. Optional: To clear the command line and start over, press CANCEL.
3. Press ENTER.

Example: If you execute this program segment

```
"Variable name?" "VAR:" INPUT »
```

the display looks like this:
Example: The following program, VSPH, calculates the volume of a sphere. VSPH prompts for the radius of the sphere, then cubes it and multiplies by \(\frac{4}{3} \pi\). VSPH executes INPUT to prompt for the radius. INPUT sets Program-entry mode when program execution pauses for data entry.

Program:```````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````
Execute \( VSPH \) to calculate the volume of a sphere of radius 2.5.

\[
\text{VAR } VSPH
\]

Key in the radius and continue program execution.

\[
2.5 \text{ ENTER}
\]

To include INPUT options:

- Use a list (with \( \xi \) \( \xi \) delimiters) as the command-line argument for INPUT. The list can contain one or more of the following:
  - Command-line string (with \( \" \) \( \" \) delimiters).
  - Cursor position as a real number or as a list containing two real numbers.
  - Operating options \( \#L, G, \alpha, \text{ or } \forall \).

In its general form, the level 1 argument for INPUT is a list that specifies the content and interpretation of the command line. The list can contain one or more of the following parameters in any order:

\[
\xi \text{" command-line" cursor-position operating-options } \xi
\]

- \( \text{" command-line"} \) Specifies the content of the command line when the program pauses. Embedded newline characters produce multiple lines in the display. (If not included, the command line is blank.)
- \( \text{cursor-position} \) Specifies the position of the cursor in the command line and its type. (If not included, an insert cursor is at the end of the command line.)
  - A \text{real number } n \text{ specifies the } n\text{th character in the first row (line) of the command line. Zero specifies the end of the command-line string. A positive number specifies the insert cursor—a negative number specifies the replace cursor.}
  - A \text{list } \xi \text{ row character } \xi \text{ specifies the row and character position. Row 1 is the first row (line) of the command line. Characters count from}
the left end of each row—character 0 specifies the end of the row. A positive row number specifies the insert cursor—a negative row number specifies the replace cursor.

**operating-options** Specify the input setup and processing using zero or more of these unquoted names:

- **ALG** activates Algebraic/Program-entry mode (for algebraic syntax). (If not included, Program-entry mode is active.)
- **α** (α (A)) specifies alpha lock. (If not included, alpha is inactive.)
- **✓** verifies whether the result string (without the " " delimiters) is a valid object or sequence of objects. If the result string isn't valid, INPUT displays the Invalid Syntax message and prompts again for data. (If not included, syntax isn't checked.)

To design the command-line string for INPUT:

- For simple input, use a string that produces a valid object:
  - Use an empty string.
  - Use a :label: tag.
  - Use a "text" comment.
- For special input, use a string that produces a recognizable pattern.

After the user enters input in the command line and presses ENTER to resume execution, the contents of the command line are returned to level 1 as the result string. The result string normally contains the original command-line string, too. If you design the command-line string carefully, you can ease the process of extracting the input data.

To process the result string from INPUT:

- For simple input, use OBJ→ to convert the string into its corresponding objects.
- For sensitive input, use the ✓ option for INPUT to check for valid objects, then use OBJ→ to convert the string into those objects.
- For special input, process the input as a string object, possibly extracting data as substrings.
Example: The program VSPH on page 1-61 uses an empty command-line string.

Example: The program SSEC on page 1-66 uses a command-line string whose characters form a pattern. The program extracts substrings from the result string.

Example: The command-line string "@UPPER LIMIT@" displays @UPPER LIMIT@ in the command line. If you press 200 ENTER, the return string is "@UPPER LIMIT200". When OBJ→ extracts the text from the string, it strips away the @ characters and the enclosed characters, and it returns the number 200. (See "Creating Programs on a Computer" on page 1-10 for more information about @ comments.)

Example: The following program, TINPUT, executes INPUT to prompt for the inner and outer radii of a torus, then calls TORSA (page 1-45) to calculate its surface area. TINPUT prompts for a and b in a two-row command line. The level 1 argument for INPUT is a list that contains:

- The command-line string, which forms the tags and delimiters for two tagged objects.
- An embedded list specifying the initial cursor position.
- The % parameter to check for invalid syntax in the result string.
**Program:**

```
< "Key in a, b"
{ "a:≡:b:" (1 0) V }
```

**Comments:**

The level 2 string, displayed at the top of the stack area.
The level 1 list contains a string, a list, and the verify option. (To key
in the string, press \(\leftarrow\) " \(\rightarrow\) " \(\rightarrow\) a \(\leftarrow\) \(\rightarrow\) \(\rightarrow\) \(\rightarrow\) \(\rightarrow\) \(\rightarrow\) b.
After you press \(\text{ENTER}\) to put
the finished program on the stack,
the string is shown on one line,
with \# indicating the newline
character.) The embedded list
puts the insert cursor at the end
of row 1.

INPUT

Displays the stack and
command-line strings, positions
the cursor, sets Program-entry
mode, and suspends execution for
input.

OBJ≠

Converts the string into its
component objects—two tagged
objects.

TORS

Calls TORS to calculate the
surface area.

STORE \(\text{ENTER} \ 1\ 	ext{TINPUT \ STO}\)
Stores the program in TINPUT.

Execute TINPUT to calculate the surface area of a torus of inner
radius \(a = 10\) and outer radius \(b = 20\).
Key in the value for $a$, press $\downarrow$ to move the cursor to the next prompt, then key in the value for $b$.

10 $\downarrow$ 20

Continue program execution.

Example: The following program executes INPUT to prompt for a social security number, then extracts two strings: the first three digits and last four digits. The level 1 argument for INPUT specifies:

- A command-line string with dashes.
- The replace cursor positioned at the start of the prompt string ($-1$). This lets the user “fill in” the command line string, using $\downarrow$ to skip over the dashes in the pattern.
- By default, no verification of object syntax—the dashes make the content invalid as objects.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>&quot;last four digits&quot;</td>
<td>&quot;first three digits&quot;</td>
</tr>
</tbody>
</table>
Program:
```
"Key in S.S. #"
{ " " " " -1 }

INPUT
DUP 1 3 SUB
SWAP
8 11 SUB
```

Comments:
- Prompt string.
- Command-line string (3 spaces before the first -, 2 spaces between, and 4 spaces after the last -).
- Suspends the program for input.
- Copies the result string, then extracts the first three and last four digits in string form.
- Stores the program in SSEC.

Using INFORM and CHOOSE for Input

You can use input forms (dialog boxes), and choose boxes for program input. Programs that contain input forms or choose boxes wait until you acknowledge them (OK or CANCEL) before they continue execution.

If OK is pressed, CHOOSE returns the selected item (or its designated returned value) to level 2 and a 1 to level 1. INFORM returns a list of field values to level 2 and a 1 to level 1.

Both the INFORM and CHOOSE commands return 0 if CANCEL is pressed.

To set up an input form:

1. Enter a title string for the input form (use \texttt{\textquotesingle\textquotesingle} or \texttt{\textquotesingle\textquotesingle}).
2. Enter a list of field specifications.
3. Enter a list of format options.
4. Enter a list of reset values (values that appear when \texttt{RESET} is pressed).
5. Enter a list of default values.
6. Execute the INFORM command.

Example: Enter a title \texttt{"FIRST ONE"} \texttt{ENTER}.
Specify a field \{ \texttt{"Name":}\} \texttt{ENTER}.
Enter format options (one column, tabs stop width five) \{ 1 5 \} \texttt{ENTER}.
Enter reset value for the field "THERESA" (ENTER).
Enter default value for the field "WENDY" (ENTER).
Execute INFORM (PRG NXT INFORM).
The screen on the left appears. Press NXT RESET OK and the screen on the right appears.

You can specify a help message and the type of data that must be entered in a field by entering field specifications as lists. For example, ("Name: " Enter your name " defaulted to: " terrestrial " accepts only object type 2 (strings) as input.

To set up a choose box:

1. Enter a title string for the choose box.
2. Enter a list of items. If this is a list of two-element lists, the first element is displayed in the choose box, and the second element is returned to level 2 when OK is pressed.
3. Enter a position number for the default highlighted item. (0 makes a view-only choose box.)
4. Execute the CHOOSE command.

Example: Enter a title "FIRST ONE" (ENTER).
Enter a list of items (ONE TWO THREE) (ENTER).
Enter a position number for default highlighted value 3 (ENTER).
Execute CHOOSE (PRG NXT INFORM). 
The following choose box appears:
Example: The following program uses input forms, choose boxes, and message boxes to create a simple phone list database.

Program:
```
' NAMES' VTYPE
  IF -1 ==
  THEN { } ' NAMES' STO
  END
WHILE
" PHONELIST OPTIONS:
{
( "ADD A NAME" 1)
( "VIEW A NUMBER" 2 )
} 1 CHOOSE
REPEAT \to c <<
CASE c 1 ==
  THEN
  WHILE
```

Comments:
Checks if the name list (NAMES) exists, if not, creates an empty one.

While cancel is not pressed, creates a choose box that lists the database options. When OK is pressed, the second item in the list pair is returned to the stack.
Stores the returned value in c.
Case 1 (ADD name), while cancel is not pressed, do the following:
Program:
"ADD A NAME"
(  "NAME:" "ENTER NAME" 2 )
( "PHONE:" "ENTER A PHONE NUMBER" 2 )
( ) ( ) ( ) INFORM
REPEAT
DUP
  IF ( NOVAL ) HEAD POS
  THEN
  DROP
  "Complete both fields before pressing OK"
  MSGBOX
ELSE 1
  "LIST NAMES + SORT
  'NAMES' STO
  END
  END
END

1 2 ==
THEN
  IF ( ) NAMES SAME
  THEN
  "YOU MUST ADD A NAME FIRST"
  MSGBOX

Comments:
Creates an input form that gets the name and phone number. The two fields accept only strings (object type 2).

Checks if either field in the new entry is blank.

If either one is, displays a message.

If neither are, adds the list to NAMES, sorts it, and stores it back in NAMES. Ends the IF structure, the WHILE loop, and the CASE statement.

Case 2 (View a Number).

Checks if NAMES is an empty list.
If it is, displays a message.
Program:
ELSE
  WHILE
    "VIEW A NUMBER"
    NAMES 1 CHOOSE
  REPEAT
    →STR MSGBOX
  END
END
END
END
>
END
>
(ENTER) [F] PHONES [STO]

Comments:
If NAMES isn’t empty,
creates a choose box using
NAMES as choice items.

When OK is pressed, the
second item in the NAMES
list pairs (the phone
number) is returned. Makes
it a string and displays it.
Ends the WHILE loop, the
IF structure, and the CASE
statement.

Ends the CASE structure,
marks the end of the local
variable defining procedure,
ends the WHILE loop, and
marks the end the program.
Stores the program in
PHONES.

You can delete names and numbers by editing the NAMES variable.
To improve upon this program, create a delete name routine.

Beeping to Get Attention

To enter BEEP in a program:

1. Enter a number that specifies the tone frequency in hertz.
2. Enter a number that specifies the tone duration in seconds.
3. Enter the BEEP command ([PRG] [NXT] [OUT] menu).

   « ... frequency duration BEEP ... »

BEEP takes two arguments from the stack: the tone frequency from
level 2 and the tone duration from level 1.

Example: The following edited version of T_PROMPT sounds a
440-hertz, one-half-second tone at the prompt for data input.
Program:
«
"ENTER a, b IN ORDER:"
440 .5 BEEP
PROMPT TORSA »

Comments:
Sounds a tone just before the prompt for data input.

Stopping a Program for Keystroke Input

A program can stop for keystroke input—it can wait for the user to press a key. You can do this with the WAIT and KEY commands.

Using WAIT for Keystroke Input

The WAIT command normally suspends execution for a specified number of seconds. However, you can specify that it wait indefinitely until a key is pressed.

To enter WAIT in a program:

- To stop without changing the display, enter 0 and the WAIT command (PRG IN menu).
- To stop and display the current menu, enter —1 and the WAIT command (PRG IN menu).

WAIT takes the 0 or —1 from level 1, then suspends execution until a valid keystroke is executed.

For an argument of —1, WAIT displays the currently specified menu. This lets you build and display a menu of user choices while the program is paused. (A menu built with MENU or TMENU is not normally displayed until the program ends or is halted.)

When execution resumes, the three-digit key location number of the pressed key is left on the stack. This number indicates the row, column, and shift level of the key.
To respond to WAIT while running a program:

- Press any valid keystroke. (A prefix key such as ↓ or → by itself is not a valid keystroke.)

**Using KEY for Keystroke Input**

You can use KEY inside an indefinite loop to “pause” execution until any key—or a certain key—is pressed.

**To enter a KEY loop in a program:**

1. Enter the loop structure.
2. In the test-clause sequence, enter the KEY command (PRG IN menu) plus any necessary test commands.
3. In the loop-clause, enter no commands to give the appearance of a “paused” condition.

KEY returns 0 to level 1 when the loop begins. It continues to return 0 until a key is pressed—then it returns 1 to level 1 and the two-digit row-column number of the pressed key to level 2. For example, ENTER returns 51, and ← returns 71.

The test-clause should normally cause the loop to repeat until a key is pressed. If a key is pressed, you can use comparison tests to check the value of the key number. (See “Using Indefinite Loop Structures” on page 1-36 and “Using Comparison Functions” on page 1-17.)

**To respond to a KEY loop while running a program:**

- Press any key. (A prefix key such as ↓ or → is a valid key.)

**Example:** The following program segment returns 1 to level 1 if ➔ is pressed, or 0 to level 1 if any other key is pressed:

```
« ... DO UNTIL KEY END 95 SAME ... »
```
Output

You can determine how a program presents its output. You can make the output more recognizable using the techniques described in this section.

Data Output Commands

<table>
<thead>
<tr>
<th>Key</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRG</td>
<td>PVIEW</td>
<td>Displays PICT starting at the given coordinates.</td>
</tr>
<tr>
<td>NXT</td>
<td>TEXT</td>
<td>Displays the stack display.</td>
</tr>
<tr>
<td></td>
<td>CLLCD</td>
<td>Blanks the stack display.</td>
</tr>
<tr>
<td></td>
<td>DISP</td>
<td>Displays an object in the specified line.</td>
</tr>
<tr>
<td></td>
<td>FREEZE</td>
<td>&quot;Freezes&quot; a specified area of the display until a key press.</td>
</tr>
<tr>
<td></td>
<td>MSGBOX</td>
<td>Creates a user-defined message box.</td>
</tr>
<tr>
<td></td>
<td>BEEP</td>
<td>Sounds a beep at a specified frequency (in hertz, level 2) and duration (in seconds, level 1).</td>
</tr>
</tbody>
</table>

Labeling Output with Tags

To label a result with a tag:

1. Put the output object on the stack.
2. Enter a tag—a string, a quoted name, or a number.
3. Enter the →TAG command (PRG TYPE menu).

```
... object tag →TAG ...
```

→TAG takes two arguments—an object and a tag—from the stack and returns a tagged object.

Example: The following program TTAG is identical to TINPUT, except that it returns the result as AREA: value.
Program:
«
"Key in a, b"
( "a:=b:" (1 0) V )
INPUT OBJ→
TORSA
"AREA"
→TAG
»

ENTER ▼ TTAG STO

Entersthe tag (a string).
Uses the program result and
string to create the tagged object.

Stores the program in TTAG.

Execute TTAG to calculate the area of a torus of inner radius $a = 1.5$ and outer radius $b = 1.85$. The answer is returned as a tagged object.

Labeling and Displaying Output as Strings

To label and display a result as a string:

1. Put the output object on the stack.
2. Enter the →STR command (PRG TYPE menu).
3. Enter a string to label the object (with " " delimiters).
4. Enter the SWAP + commands to swap and concatenate the strings.
5. Enter a number specifying the line to display the string on.
6. Enter the DISP command (PRG OUT menu).

« ... object →STR label SWAP + line DISP ... »

DISP displays a string without its " " delimiters.

Example: The following program TSTRING is identical to TINPUT, except that it converts the program result to a string and appends a labeling string to it.
Program:

«
"Key in a, b"
{ "a:≡:b:" (1 0) V }
INPUT OBJ→
TORSA
→STR
"Area = "
SWAP +
CLLCD 1 DISP 1 FREEZE
»

(ENTER) [ TSTRING STO]

Comments:

Converting the result to a string.

Enters the labeling string.

Swaps and adds the two strings.

Displays the resultant string, without its delimiters, in line 1 of the display.

Stores the program in TSTRING.

Execute TSTRING to calculate the area of the torus with \( a = 1.5 \) and \( b = 1.85 \). The labeled answer appears in the status area.

To pause to display a result:

1. Enter commands to set up the display.
2. Enter the number of seconds you want to pause.
3. Enter the WAIT command (PRG IN menu).

WAIT suspends execution for the number of seconds in level 1. You can use WAIT with DISP to display messages during program execution—for example, to display intermediate program results. (WAIT interprets arguments 0 and -1 differently—see “Using WAIT for Keystroke Input” on page 1-72.)
Using MSGBOX to Display Output

To set up a message box:

1. Enter a message string.
2. Execute the MSGBOX command.

**Example:** Enter a string "HELLO, WORLD" (ENTER).
Execute MSGBOX \(\text{PRG} \quad \text{NXT} \quad \text{OUT} \quad \text{MSGBOX}\).
The following message appears:

You must acknowledge a message box by pressing \(\text{OK}\) or \(\text{CANCEL}\).

Using Menus with Programs

You can use menus with programs for different purposes:

- **Menu-based input.** A program can set up a menu to get input during a halt in a program—then resume executing the same program.
- **Menu-based application.** A program can set up a menu and finish executing, leaving the menu to start executing other related programs.

To set up a built-in or library menu:

1. Enter the menu number.
2. Enter the MENU command (MODES MENU menu).
To set up a custom menu:

1. Enter a list (with \( \{ \} \) delimiters) or the name of a list defining the menu actions. If a list of two element lists is given, the first element appears in the menu, but it is the second element that is returned to the stack when the menu key is pressed.

2. Activate the menu:
   - To save the menu as the CST menu, enter the MENU command (MODES MENU menu).
   - To make the menu temporary, enter the TMENU command (MODES MENU menu).

The menu isn’t displayed until program execution halts.

Menu numbers for built-in menus are listed in chapter 3, under the MENU command. Library menus also have numbers—the library number serves as the menu number. So you can activate applications menus (such as the SOLVE and PLOT menus) and other menus (such as the VAR and CST menus) in programs. The menus behave just as they do during normal keyboard operations.

You create a custom menu to cause the behavior you need in your program—see the topics that follow. You can save the menu as the CST menu, so the user can get it again by pressing [CST]. Or you can make it temporary—it remains active (even after execution stops), but only until a new menu is selected—and it doesn’t affect the contents of variable CST.

To specify a particular page of a menu, enter the number as \( m.pp \), where \( m \) is the menu number and \( pp \) is the page number (such as 94.02 for page 2 of the TIME menu). If page \( pp \) doesn’t exist, page 1 is displayed (94 gives page 1 of the TIME menu).

**Example:** Enter 69 MENU to get page 1 of the MODES MISC menu. Enter 69.02 MENU to get page 2 of the MODES MISC menu.

**To restore the previous menu:**
- Execute 0 MENU.

**To recall the menu number for the current menu:**
- Execute the RCLMENU command (MODES MENU menu).
Using Menus for Input

To display a menu for input in a program:

1. Set up the menu—see the previous section.
2. Enter a command sequence that halts execution (such as DISP, PROMPT, or HALT).

The program remains halted until it’s resumed by a CONT command, such as by pressing \[\text{CONT}\]. If you create a custom menu for input, you can include a CONT command to automatically resume the program when you press the menu key.

Example: The following program activates page 1 of the MODES ANGL menu and prompts you to set the angle mode. After you press the menu key, you have to press \[\text{CONT}\] to resume execution.

```
< 65 MENU "Select Angle Mode" PROMPT >
```

Example: The PIE program on page 2-49 assigns the CONT command to one key in a temporary menu.

Example: The MNX program on page 2-22 sets up a temporary menu that includes a program containing CONT to resume execution automatically.

Using Menus to Run Programs

You can use a custom menu to run other programs. That menu can serve as the main interface for an application (a collection of programs).

To create a menu-based application:

1. Create a custom menu list for the application that specifies programs as menu objects.
2. Optional: Create a main program that sets up the application menu—either as the CST menu or as a temporary menu.

Example: The following program, WGT, calculates the mass of an object in either English or SI units given the weight. WGT displays a temporary custom menu, from which you run the appropriate program. Each program prompts you to enter the weight in the desired unit system, then calculates the mass. The menu remains...
active until you select a new menu, so you can do as many calculations as you want.

Enter the following list and store it in LST:
{
  "ENGL" « "ENTER Wt in POUNDS" PROMPT 32.2 / »
  "SI" « "ENTER Wt in NEWTONS" PROMPT 9.81 / »
}

Program:  
« LST TMENU »

Comments:  
Displays the custom menu stored in LST.
Stores the program in WGT.

Use WGT to calculate the mass of an object of weight 12.5 N. The program sets up the menu, then completes execution.

Select the SI unit system, which starts the program in the menu list.

Key in the weight, then resume the program.

Example: The following program, EIZ, constructs a custom menu to emulate the HP Solve application for a capacitive electrical circuit. The program uses the equation \( E = IZ \), where \( E \) is the voltage, \( I \) is the current, and \( Z \) is the impedance.

Because the voltage, current, and impedance are complex numbers, you can't use the HP Solve application to find solutions. The custom menu in EIZ assigns a direct solution to the left-shifted menu key for each variable, and assigns store and recall functions to the unshifted and right-shifted keys—the actions are analogous to the HP Solve
application. The custom menu is automatically stored in CST, replacing the previous custom menu—you can press CST to restore the menu.

**Program:**

```
"DEG -15 SF -16 SF
2 FIX

{
"E" « « 'E' STO »
"I Z * DUP 'E' STO
"E: " SWAP + CLLCD
1 DISP 1 FREEZE »
"E » ) }

{
"I" « « 'I' STO »
"E Z / DUP 'I' STO
"I: " SWAP + CLLCD
1 DISP 1 FREEZE »
"I » ) }

{
"Z" « « 'Z' STO »
"E I / DUP 'Z' STO
"Z: " SWAP + CLLCD
1 DISP 1 FREEZE »
"Z » ) }

) } MENU

ENTER ['] EIZ STO
```

**Comments:**

Sets Degrees mode. Sets flags -15 and -16 to display complex numbers in polar form. Sets the display mode to 2 Fix. Starts the custom menu list. Builds menu key 1 for E.

Unshifted action: stores the object in E. Left-shift action: calculates I x Z, stores it in E, and displays it with a label. Right-shift action: recalls the object in E. Builds menu key 2.

Builds menu key 3.

Ends the list. Displays the custom menu. Stores the program in EIZ.

For a 10-volt power supply at phase angle 0°, you measure a current of 0.37-amp at phase angle 68°. Find the impedance of the circuit using EIZ.

[ENTER] CLEAR VAR EIZ
Key in the voltage value.

\[ 10 \triangle 0 \]

Store the voltage value. Then key in and store the current value. Solve for the impedance.

\[ .37 \triangle 68 \]

Recall the current and double it. Then find the voltage.

\[ 2 \times I \]

Press \([\text{MODES}]\) \(\text{FMT} \) \(\text{STD} \) and \([\text{NXT}] \) \(\text{MODE} \) \(\text{ANGL} \) \(\text{RECT} \) to restore Standard and Rectangular modes.

### Turning Off the HP 48 from a Program

**To turn off the calculator in a program:**

- Execute the OFF command (PRG RUN menu).

The OFF command turns off the HP 48. If a program executes OFF, the program resumes when the calculator is next turned on.
Programming Examples

The programs in this chapter demonstrate basic programming concepts. These programs are intended to improve your programming skills, and to provide supplementary functions for your calculator.

At the end of each program, the program's checksum and size in bytes are listed to help make sure you typed the program in correctly. (The checksum is a binary integer that uniquely identifies the program based on its contents). To make sure you've keyed the program in correctly, store it in its name, put the name in level 1, then execute the BYTES command ([EQU (MEMORY) BYTES]). This returns the program's checksum to level 2, and its size in bytes to level 1. (If you execute BYTES with the program object in level 1, you'll get a different byte count.)

The programs in this chapter are also included in the online information of the Program Development Link software for developing HP 48 programs on computers. This software lets you load these programs from the online information into your HP 48 through its serial port.

The examples in this chapter assume the HP 48 is in its initial, default condition—they assume you haven’t changed any of the HP 48 operating modes. (To reset the calculator to this condition, see “Memory Reset” in chapter 5 of the HP 48 User’s Guide.)

Each program listing in this chapter gives the following information:

- A brief description of the program.
- A syntax diagram (where needed) showing the program’s required inputs and resulting outputs.
- Discussion of special programming techniques in the program.
- Any other programs needed.
- The program listing.
- The program’s checksum and byte size.
Fibonacci Numbers

This section includes three programs that calculate Fibonacci numbers:

- **FIB1** is a user-defined function that is defined *recursively* (that is, its defining procedure contains its own name). **FIB1** is short.
- **FIB2** is a user-defined function with a definite loop. It’s longer and more complicated than **FIB1**, but faster.
- **FIBT** calls both **FIB1** and **FIB2** and calculates the execution time of each subprogram.

**FIB1** and **FIB2** demonstrate an approach to calculating the $n$th Fibonacci number $F_n$, where:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

**FIB1 (Fibonacci Numbers, Recursive Version)**

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>→</td>
<td>$F_n$</td>
</tr>
</tbody>
</table>

**Techniques used in FIB1**

- **IFTE (if-then-else function).** The defining procedure for **FIB1** contains the conditional function **IFTE**, which can take its argument either from the stack or in algebraic syntax.
- **Recursion.** The defining procedure for **FIB1** is written in terms of **FIB1**, just as $F_n$ is defined in terms of $F_{n-1}$ and $F_{n-2}$. 
FIB1 program listing

Program:

\[ \begin{align*}
& \ \rightarrow \ n \\
& ' \text{IFTE}(n \leq 1, \\
& \quad n, \\
& \quad \text{FIB1}(n-1) + \text{FIB1}(n-2))' \\
& \end{align*} \]

Comments:

Defines local variable \( n \).
The defining procedure, an algebraic expression. If \( n \leq 1 \), \( F_n = n \), else \( F_n = F_{n-1} + F_{n-2} \).

Stores the program in \( \text{FIB1} \).

Checksum: \# 41467d (press \( \text{FIB1} \) \( \text{FIB1} \) \( \text{MEMORY} \) \( \text{BYTES} \))
Bytes: 113.5

Example: Calculate \( F_6 \). Calculate \( F_{10} \) using algebraic syntax.

First calculate \( F_6 \).

\[ \begin{align*}
& \text{VAR} \\
& 6 \ \text{FIB1} \\
& \end{align*} \]

Next, calculate \( F_{10} \) using algebraic syntax.

\[ \begin{align*}
& ' \text{FIB1} (10) \text{EVAL} \\
& \end{align*} \]

FIB2 (Fibonacci Numbers, Loop Version)

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( \rightarrow )</td>
<td>( F_n )</td>
</tr>
</tbody>
</table>

Techniques used in FIB2

- \( \text{IF} \ldots \text{THEN} \ldots \text{ELSE} \ldots \text{END} \). \( \text{FIB2} \) uses the program-structure form of the conditional. (\( \text{FIB1} \) uses IFTE.)
START ... NEXT (definite loop). To calculate $F_n$, $FIB2$ starts with $F_0$ and $F_1$ and repeats a loop to calculate successive values of $F_i$.

**FIB2 program listing**

Program:

```
\> n
\> IF n \leq
1 THEN n
ELSE
0 1
2 n
START
DUP
ROT
+
NEXT
SWAP DROP
END
```

Comments:

- Creates a local variable structure.
- If $n \leq 1$,
- then $F_n = n$;
- otherwise ...
- Puts $F_0$ and $F_1$ on the stack.
- From 2 to $n$ does the following loop:
- Copies the latest $F$ (initially $F_1$).
- Gets the previous $F$ (initially $F_0$).
- Calculates the next $F$ (initially $F_2$).
- Repeats the loop.
- Drops $F_{n-1}$.
- Ends the ELSE clause.
- Ends the defining procedure.
- Stores the program in $FIB2$.

Checksum: # 51820d (press \FIB2 \STO\) MEMOY BYTES)
Bytes: 89

Example: Calculate $F_6$ and $F_{10}$.

Calculate $F_6$.

```
VAR
6 FIB2
```

2-4 Programming Examples
FIBT (Comparing Program-Execution Time)

FIB1 calculates intermediate values \( F_i \) more than once, while FIB2 calculates each intermediate \( F_i \) only once. Consequently, FIB2 is faster. The difference in speed increases with the size of \( n \) because the time required for FIB1 grows exponentially with \( n \), while the time required for FIB2 grows only linearly with \( n \).

FIBT executes the TICKS command to record the execution time of FIB1 and FIB2 for a given value of \( n \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( F_n )</td>
<td>FIB1 TIME: ( z )</td>
<td>FIB2 TIME: ( z )</td>
</tr>
</tbody>
</table>

Techniques used in FIBT

- **Structured programming.** FIBT calls both FIB1 and FIB2.
- **Programmatic use of calculator clock.** FIBT executes the TICKS command to record the start and finish of each subprogram.
- **Labeling output.** FIBT tags each execution time with a descriptive message.

Required Programs

- **FIB1** (page 2-2) calculates \( F_n \) using recursion.
- **FIB2** (page 2-3) calculates \( F_n \) using looping.
FIBT program listing

Program:
```
DUP TICKS SWAP FIB1
SWAP TICKS SWAP
- B→R 8192 /

"FIB1 TIME" →TAG
ROT TICKS SWAP FIB2
TICKS
SWAP DROP SWAP
- B→R 8192 /

"FIB2 TIME" →TAG
```

Comments:
Copies n, then executes FIB1, recording the start and stop time. Calculates the elapsed time, converts it to a real number, and converts that number to seconds. Leaves the answer returned by FIB1 in level 2. Tags the execution time.

Executes FIB2, recording the start and stop time. Drops the answer returned by FIB2 (FIB1 returned the same answer). Calculates the elapsed time for FIB2 and converts to seconds. Tags the execution time.

Stores the program in FIBT.

Checksum: # 22248d
Bytes: 135

Example: Calculate F_{13} and compare the execution time for the two methods.

Select the VAR menu and do the calculation.

```
13 FIBT
```

F_{13} is 233. FIB2 takes fewer seconds to execute than FIB1 (far fewer if n is large). (The times required for the calculations depend on the
Displaying a Binary Integer

This section contains three programs:

- **PAD** is a utility program that converts an object to a string for right-justified display.
- **PRESERVE** is a utility program for use in programs that change the calculator's status (angle mode, binary base, and so on).
- **BDISP** displays a binary integer in HEX, DEC, OCT, and BIN bases. It calls **PAD** to show the displayed numbers right-justified, and it calls **PRESERVE** to preserve the binary base.

**PAD (Pad with Leading Spaces)**

**PAD** converts an object to a string, and if the string contains fewer than 22 characters, adds spaces to the beginning of the string till the string reaches 22 characters.

When a short string is displayed with DISP, it appears **left-justified**: its first character appears at the left end of the display. By adding spaces to the beginning of a short string, **PAD** moves the string to the right. When the string (including leading spaces) reaches 22 characters, it appears **right-justified**: its last character appears at the right end of the display. **PAD** has no effect on longer strings.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>—</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>→</td>
<td>&quot; object&quot;</td>
</tr>
</tbody>
</table>

Techniques used in PAD

- **WHILE ... REPEAT ... END** (indefinite loop). The WHILE clause contains a test that executes the REPEAT clause and tests again (if true) or skips the REPEAT clause and exits (if false).
- **String operations.** *PAD* demonstrates how to convert an object to string form, count the number of characters, and combine two strings.

**PAD program listing**

Program:
```
<<
→STR

WHILE
  DUP SIZE 22 <
REPEAT
  " " SWAP +
END
>
```

Checksum: # 38912d
Bytes: 61.5

*PAD* is demonstrated in the program *BDISP*.

**PRESERVE (Save and Restore Previous Status)**

*PRESERVE* stores the current calculator (flag) status, executes a program from the stack, and restores the previous status.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;&lt; program &gt;&gt;</code></td>
<td>→</td>
<td><code>result of program</code></td>
</tr>
<tr>
<td><code>'program name'</code></td>
<td>→</td>
<td><code>result of program</code></td>
</tr>
</tbody>
</table>
Techniques used in PRESERVE

- **Preserving calculator flag status.** *PRESERVE* uses RCLF (*recall flags*) to record the current status of the calculator in a binary integer, and STOF (*store flags*) to restore the status from that binary integer.

- **Local-variable structure.** *PRESERVE* creates a local variable structure to briefly remove the binary integer from the stack. Its defining procedure simply evaluates the program argument, then puts the binary integer back on the stack and executes STOF.

- **Error trapping.** *PRESERVE* uses IFERR to trap faulty program execution on the stack and to restore flags. DOERR shows the error if one occurs.

**PRESERVE program listing**

**Program:**

```plaintext
Recalls the list of two 64-bit binary integers representing the status of the 64 system flags and 64 user flags.
```

```plaintext
Stores the list in local variable f.
```

```plaintext
Begins the defining procedure.
```

```plaintext
Starts the error trap.
```

```plaintext
Executes the program placed on the stack as the level 1 argument.
```

```plaintext
If the program caused an error, restores flags, shows the error, and aborts execution.
```

```plaintext
Ends the error routine.
```

```plaintext
Puts the list back on the stack, then restores the status of all flags.
```

```plaintext
Ends the defining procedure.
```

```plaintext
Stores the program in PRESERVE.
```

**Comments:**

- Recalls the list of two 64-bit binary integers representing the status of the 64 system flags and 64 user flags.
- Stores the list in local variable f.
- Begins the defining procedure.
- Starts the error trap.
- Executes the program placed on the stack as the level 1 argument.
- If the program caused an error, restores flags, shows the error, and aborts execution.
- Ends the error routine.
- Ends the defining procedure.
- Stores the program in PRESERVE.
PRESERVE is demonstrated in the program BDISP.

BDISP (Binary Display)

BDISP displays a real or binary number in HEX, DEC, OCT, and BIN bases.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td># n</td>
<td>→</td>
<td># n</td>
</tr>
<tr>
<td>n</td>
<td>→</td>
<td>n</td>
</tr>
</tbody>
</table>

Techniques used in BDISP

- **IFERR . . . THEN . . . END (error trap).** To accommodate real-number arguments, BDISP includes the command R→B (*real-to-binary*). However, this command causes an error if the argument is *already* a binary integer. To maintain execution if an error occurs, the R→B command is placed inside an IFERR clause. No action is required when an error occurs (since a binary number is an acceptable argument), so the THEN clause contains no commands.

- **Enabling LASTARG.** In case an error occurs, the LASTARG recovery feature must be enabled to return the argument (the binary number) to the stack. BDISP clears flag -55 to enable this.

- **FOR . . . NEXT loop (definite loop with counter).** BDISP executes a loop from 1 to 4, each time displaying n (the number) in a different base on a different line. The loop counter (named j in this program) is a local variable created by the FOR . . . NEXT program structure (rather than by a + command), and automatically incremented by NEXT.

- **Unnamed programs as arguments.** A program defined only by its « and » delimiters (not stored in a variable) is not automatically evaluated, but is placed on the stack and can be used as an

2-10 Programming Examples
argument for a subroutine. \textit{BDISP} demonstrates two uses for unnamed program arguments:

- \textit{BDISP} contains a main program argument and a call to \textit{PRESERVE}. This program argument goes on the stack and is executed by \textit{PRESERVE}.
- \textit{BDISP} also contains four program arguments that "customize" the action of the loop. Each of these contains a command to change the binary base, and each iteration of the loop evaluates one of these arguments.

When \textit{BDISP} creates a local variable for \textit{n}, the defining procedure is an unnamed program. However, since this program is a defining procedure for a local variable structure, it is automatically executed.

\textbf{Required Programs}

- \textit{PAD} (page 2-7) expands a string to 22 characters so that \textit{DISP} shows it right-justified.
- \textit{PRESERVE} (page 2-8) stores the current status, executes the main nested program, and restores the status.

\textbf{BDISP program listing}

<table>
<thead>
<tr>
<th>Program:</th>
<th>Comments:</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>Begins the main nested program.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Makes a copy of \textit{n}.</td>
</tr>
<tr>
<td>DUP</td>
<td>Clears flag \textit{-55} to enable \textit{LASTARG}.</td>
</tr>
<tr>
<td>\textit{-55 CF}</td>
<td>Begins error trap.</td>
</tr>
<tr>
<td>\textit{IFERR}</td>
<td>Converts \textit{n} to a binary integer.</td>
</tr>
<tr>
<td>\textit{R#B}</td>
<td>If an error occurs, do nothing (no commands in the \textit{THEN} clause).</td>
</tr>
<tr>
<td>\textit{THEN}</td>
<td>Creates a local variable \textit{n} and begins the defining program.</td>
</tr>
<tr>
<td>\textit{END}</td>
<td>Clears the display.</td>
</tr>
<tr>
<td>\textit{→ n}</td>
<td>Nested program for \textit{BIN}.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Nested program for \textit{OCT}.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Nested program for \textit{DEC}.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Nested program for \textit{HEX}.</td>
</tr>
</tbody>
</table>

\textit{Programming Examples 2-11}
Program:

\[
\begin{align*}
1 & 4 \\
\text{FOR } j & \\
\text{EVAL} & \\
\rightarrow & \text{STR} \\
\text{PAD} & \\
j & \text{DISP} \\
\text{NEXT} & \\
\end{align*}
\]

Comments:

Sets the counter limits.
Starts the loop with counter \( j \).
Executes one of the nested base programs (initially for HEX).
Makes a string showing \( n \) in the current base.
Pads the string to 22 characters.
Displays the string in the \( j \)th line.
Increments \( j \) and repeats the loop.
Ends the defining program.
Freezes the status and stack areas.
Ends the main nested program.
Stores the current flag status, executes the main nested program, and restores the status.

Stores the program in \( BDISP \).

Checksum: \# 18055d
Bytes: 191

Example: Switch to DEC base, display \#100 in all bases, and check that \( BDISP \) restored the base to DEC.

Clear the stack and select the MTH BASE menu. Make sure the current base is DEC and enter \# 100.
Execute `BDISP`.

```
VAR BDISP
```

Return to the normal stack display and check the current base.

```
CANCEL
```

```
MTH BASE
```

Although the main nested program left the calculator in BIN base, `PRESERVE` restored DEC base. To check that `BDISP` also works for real numbers, try 144.

```
VAR
144 BDISP
```

```
# 64h
# 100d
# 144o
# 1100100b
```

```
# 90h
# 144d
# 228o
# 10010000b
```

Press `CANCEL` to return to the stack display.
Median of Statistics Data

This section contains two programs:

- \%TILE returns the value of a specified percentile of a list.
- MEDIAN uses \%TILE to calculate the median of the current statistics data.

(\%TILE and MEDIAN are included in the TEACH function’s EXAMPLES directory. See the entry for TEACH in chapter 3.)

\%TILE (Percentile of a List)

\%TILE sorts a list, then returns the value of a specified percentile of the list. For example, typing \{ list \} 50 and pressing \%TILE returns the median (50th percentile) of the list.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ list }</td>
<td>n</td>
<td>→</td>
<td>n&lt;sup&gt;th&lt;/sup&gt; percentile of sorted list</td>
</tr>
</tbody>
</table>

Techniques used in \%TILE

- FLOOR and CEIL. For an integer, FLOOR and CEIL both return that integer; for a noninteger, FLOOR and CEIL return successive integers that bracket the noninteger.
- SORT. The SORT command sorts the list elements into ascending order.
%TILE program listing

Program:

*          
SWAP SORT
DUP SIZE  
1 + ROT 100 / *
→ p
  «         
  DUP  
p FLOOR GET
  SWAP  
p CEIL GET
  + 2 /   
»         
»

(ENTER) 0 %TILE STO

Comments:

Brings the list to level 1 and sorts it.
Copies the list, then finds its size.
Calculates the position of the specified percentile.
Stores the center position in local variable p.
Begins the defining procedure.
Makes a copy of the list.
Gets the number at or below the center position.
Moves the list to level 1.
Gets the number at or above the center position.
Calculates the average of the two numbers.
Ends the defining procedure.

Stores the program in %TILE.

Checksum: # 42718d
Bytes: 99

Example: Calculate the median of the list \{8 3 1 5 2\}.

9(ENTER) 8 3 1 5 2 ENTER
VAR 50 %TILE

Programming Examples 2-15
MEDIAN (Median of Statistics Data)

MEDIAN returns a vector containing the medians of the columns of the statistics data. Note that for a sorted list with an odd number of elements, the median is the value of the center element; for a list with an even number of elements, the median is the average value of the elements just above and below the center.

Techniques used in MEDIAN

- Arrays, lists, and stack elements. MEDIAN extracts a column of data from \( \Sigma DAT \) in vector form. To convert the vector to a list, MEDIAN puts the vector elements on the stack and combines them into a list. From this list the median is calculated using \( \%TILE \).

The median for the \( m \)th column is calculated first, and the median for the first column is calculated last. As each median is calculated, ROLLD is used to move it to the top of the stack.

After all medians are calculated and positioned on the stack, they’re combined into a vector.

- FOR...NEXT (definite loop with counter). MEDIAN uses a loop to calculate the median of each column. Because the medians are calculated in reverse order (last column first), the counter is used to reverse the order of the medians.

Required Program

- \( \%TILE \) (page ) sorts a list and returns the value of a specified percentile.
Programming Examples 2-17
Program:

\[
\begin{align*}
&\text{\(m \rightarrow \text{ARRY}\)} \\
&\leq \text{STO} \Sigma \\
&\text{\(\rightarrow\)} \\
&\text{\(\text{ENTER} \ 1 \ \text{MEDIAN} \ \text{STO}\)}
\end{align*}
\]

Comments:
Combines all the medians into an \(m\)-element vector.
Restores \(\Sigma \text{DAT}\) to its previous value.
Ends the defining procedure.
Stores the program in \(\text{MEDIAN}\).

Checksum: 57504d
Bytes: 140

Example: Calculate the median of the following data.

\[
\begin{bmatrix}
18 & 12 \\
4 & 7 \\
3 & 2 \\
11 & 1 \\
31 & 48 \\
20 & 17
\end{bmatrix}
\]

There are two columns of data, so \(\text{MEDIAN}\) will return a two-element vector.

Enter the matrix.

\[
\begin{align*}
&\text{\(\text{MATRIX}\)} \\
&18 \ \text{ENTER} \ 12 \ \text{ENTER} \ \text{\(\downarrow\)} \\
&4 \ \text{ENTER} \ 7 \ \text{ENTER} \\
&3 \ \text{ENTER} \ 2 \ \text{ENTER} \\
&11 \ \text{ENTER} \ 1 \ \text{ENTER} \\
&31 \ \text{ENTER} \ 48 \ \text{ENTER} \\
&20 \ \text{ENTER} \ 17 \ \text{ENTER} \\
&\text{\(\text{ENTER}\)}
\end{align*}
\]

Store the matrix in \(\Sigma \text{DAT}\), and calculate the median.

\[
\begin{align*}
&\text{\(\text{STAT} \ \text{DATA}\)} \\
&\text{\(\text{CLE}\)} \Sigma + \\
&\text{\(\text{VAR} \ \text{MEDIA}\)}
\end{align*}
\]

\[
\begin{bmatrix}
14.5 & 9.5
\end{bmatrix}
\]
Expanding and Collecting Completely

This section contains two programs:

- \textit{MULTI} repeats a program until the program has no effect on its argument.
- \textit{EXCO} calls \textit{MULTI} to completely expand and collect an algebraic.

\textbf{MULTI (Multiple Execution)}

Given an object and a program that acts on the object, \textit{MULTI} applies the program to the object repeatedly until the program no longer changes the object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>—</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{object}</td>
<td>\langle \textit{program} \rangle</td>
<td>—</td>
<td>\textit{object}_{\text{result}}</td>
</tr>
</tbody>
</table>

Techniques used in MULTI

- \textbf{DO} \ldots \textbf{UNTIL} \ldots \textbf{END} (indefinite loop). The DO clause contains the steps to be repeated. The UNTIL clause contains the test that repeats both clauses again (if false) or exits (if true).

- Programs as arguments. Although programs are commonly named and then executed by calling their names, programs can also be put on the stack and used as arguments to other programs.

- Evaluation of local variables. The program argument to be executed repeatedly is stored in a local variable.

It’s convenient to store an object in a local variable when you don’t know beforehand how many copies you’ll need. An object stored in a local variable is simply put on the stack when the local variable is evaluated. \textit{MULTI} uses the local variable name to put the program argument on the stack and then executes EVAL to execute the program.
MULTI program listing

Program:
\[
\begin{align*}
&\times \\
&\rightarrow p \\
&\langle \\
&\text{DO} \\
&DUP \\
&p \text{ EVAL} \\
&DUP \\
&\text{ROT} \\
&\text{UNTIL SAME} \\
&\text{END} \\
&\rangle \\
&\ast \\
&\text{ENTER} \quad \text{MULTI} \quad \text{STO}
\end{align*}
\]

Comments:

- Creates a local variable \( p \) that contains the program from level 1.
- Begins the defining procedure.
- Begins the DO loop clause.
- Makes a copy of the object, now in level 1.
- Applies the program to the object, returning its new version.
- Makes a copy of the new object.
- Moves the old version to level 1.
- Begins the DO test clause.
- Tests whether the old version and the new version are the same.
- Ends the DO structure.
- Ends the defining procedure.
- Stores the program in \textit{MULTI}.

Checksum: \# 34314d
Bytes: 56

\textit{MULTI} is demonstrated in the next programming example.

**EXCO (Expand and Collect Completely)**

\textit{EXCO} repeatedly executes \textit{EXPAN} on an algebraic until the algebraic doesn't change, then repeatedly executes \textit{COLCT} until the algebraic doesn't change. In some cases the result will be a number.

Expressions with many products of sums or with powers can take many iterations of \textit{EXPAN} to expand completely, resulting in a long execution time for \textit{EXCO}.
Techniques used in EXCO

- **Subroutines.** EXCO calls the program MULTI twice. It is more efficient to create program MULTI and simply call its name twice than write each step in MULTI two times.

Required Programs

- MULTI (page 2-19) repeatedly executes the programs that EXCO provides as arguments.

EXCO program listing

Program:

```
< EXPAN >
MULTI
< COLCT >
MULTI
```

Comments:

- Puts a program on the stack as the level 1 argument for MULTI. The program executes the EXPAN command.
- Executes EXPAN until the algebraic object doesn’t change. Puts another program on the stack for MULTI. The program executes the COLCT command. Executes COLCT until the algebraic object doesn’t change.
- Stores the program in EXCO.

Checksum: # 48008d
Bytes: 65.5
Example: Expand and collect completely the expression:

\[ 3x(4y + z) + (8x - 5z)^2 \]

Enter the expression.

\[ 3 \times (4 \times Y + Z) (8 \times X - 5 \times Z) \]

Select the VAR menu and start the program.

Minimum and Maximum Array Elements

This section contains two programs that find the minimum or maximum element of an array:

- \( MNX \) uses a DO ...UNTIL ... END (indefinite) loop.
- \( MNX2 \) uses a FOR ... NEXT (definite) loop.

\( MNX \) (Minimum or Maximum Element—Version 1)

\( MNX \) finds the minimum or maximum element of an array on the stack.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([ [ \text{array} ] ])</td>
<td>([ [ \text{array} ] ])</td>
<td>(z_{\text{min}} ) or (z_{\text{max}})</td>
</tr>
</tbody>
</table>
Techniques used in MNX

- **DO ... UNTIL ... END (indefinite loop).** The DO clause contains the sort instructions. The UNTIL clause contains the system-flag test that determines whether to repeat the sort instructions.

- **User and system flags for logic control:**
  - *User flag* 10 defines the sort: When flag 10 is set, MNX finds the maximum element; when flag 10 is clear, it finds the minimum element. *You* determine the state of flag 10 at the beginning of the program.
  - *System flag* —64, the Index Wrap Indicator flag, determines when to end the sort. While flag —64 is clear, the sort loop continues. When the index invoked by GETI wraps back to the first array element, flag —64 is *automatically* set, and the sort loop ends.

- **Nested conditional.** An IF ... THEN ... END conditional is nested in the DO ... UNTIL ... END conditional, and determines the following:
  - Whether to maintain the current minimum or maximum element, or make the current element the new minimum or maximum.
  - The sense of the comparison of elements (either < or >) based on the status of flag 10.

- **Custom menu.** MNX builds a custom menu that lets you choose whether to sort for the minimum or maximum element. Key 1, labeled **MAX**, sets flag 10. Key 2, labeled **MIN**, clears flag 10.

- **Logical function.** MNX executes XOR (exclusive OR) to test the combined state of the relative value of the two elements and the status of flag 10.
MNX program listing

Program:

```
<< "MAX"
<< 10 SF CONT »
<< "MIN"
<< 10 CF CONT »

TMENU
"Sort for MAX or MIN?" PROMPT
1 GETI
DO
  ROT ROT GETI
4 ROLL DUP2
  IF
    > 10 FS? XOR
    THEN
      SWAP
      END
  DROP
UNTIL
-64 FS?
END
SWAP DROP 0 MENU
>>

(ENTER ' MNX STO)
```

Comments:

Defines the option menu. "MAX" sets flag 10 and continues execution. "MIN" clears flag 10 and continues execution.

Displays the temporary menu and a prompt message.

Gets the first element of the array. Begins the DO loop.

Puts the index and the array in levels 1 and 2, then gets the new array element.

Moves the current minimum or maximum array element from level 4 to level 1, then copies both.

Tests the combined state of the relative value of the two elements and the status of flag 10.

If the new element is either less than the current maximum or greater than the current minimum, swaps the new element into level 1.

Drops the other element off the stack.

Begins the DO test-clause.

Tests if flag -64 is set—if the index reached the end of the array.

Ends the DO loop.

Swaps the index to level 1 and drops it. Restores the last menu.

Stores the program in MNX.
Example: Find the maximum element of the following matrix:

\[
\begin{bmatrix}
12 & 56 \\
45 & 1 \\
9 & 14
\end{bmatrix}
\]

Enter the matrix.

Select the VAR menu and execute `MNX`.

Find the maximum element.

**MNX2 (Minimum or Maximum Element—Version 2)**

Given an array on the stack, `MNX2` finds the minimum or maximum element in the array. `MNX2` uses a different approach than `MNX`: it executes `OBJ→` to break the array into individual elements on the stack for testing, rather than executing `GETI` to index through the array.
Techniques used in MNX2

- **FOR ... NEXT (definite loop).** The initial counter value is 1. The final counter value is $nm - 1$, where $nm$ is the number of elements in the array. The loop-clause contains the sort instructions.

- **User flag for logic control.** *User* flag 10 defines the sort: When flag 10 is set, *MNX2* finds the maximum element; when flag 10 is clear, it finds the minimum element. *You* determine the status of flag 10 at the beginning of the program.

- **Nested conditional.** An IF ... THEN ... END conditional is nested in the FOR ... NEXT loop, and determines the following:
  - Whether to maintain the current minimum or maximum element, or make the current element the new minimum or maximum.
  - The sense of the comparison of elements (either $<$ or $>$) based on the status of flag 10.

- **Logical function.** *MNX2* executes XOR (*exclusive OR*) to test the combined state of the relative value of the two elements and the status of flag 10.

- **Custom menu.** *MNX2* builds a custom menu that lets you choose whether to sort for the minimum or maximum element. Key 1, labeled **MAX**, sets flag 10. Key 2, labeled **MIN**, clears flag 10.
MNX2 program listing

Program:
«
(C "MAX"
< 10 SF CONT >
(C "MIN"
< 10 CF CONT >)
)

TMENU
"Sort for MAX or MIN?"
PROMPT
DUP OBJ→

1
SWAP OBJ→

DROP * 1 –

FOR n
DUP2

IF
> 10 FS? XOR

THEN
SWAP
END

Comments:

Defines the temporary option menu. _MAX_ sets flag 10 and
continues execution. _MIN_ clears flag 10 and continues
execution.
Displays the temporary menu
and a prompting message.

Copies the array. Returns the
individual array elements to
levels 2 through nm+1, and
returns the list containing n
and m to level 1.
Sets the initial counter value.
Converts the list to individual
elements on the stack.
Drops the list size, then
 calculates the final counter
value (nm – 1).
Starts the FOR . . . NEXT
loop.
Saves the array elements to be
tested (initially the last two
elements). Uses the last array
element as the current
minimum or maximum.
Tests the combined state of
the relative value of the two
elements and the status of flag
10.
If the new element is either
less than the current
maximum or greater than the
current minimum, swaps the
new element into level 1.
Program:  
DROP  
NEXT  
0 MENU  

Comments:  
DROF Drops the other element off the stack.  
HEXT Ends the FOR ... NEXT loop.  
MEHL Restores the last menu.  

Stores the program in $MNX2$.  

Checksum:  \# 12277d  
Bytes:  200.5  

Example: Use $MNX2$ to find the minimum element of the matrix from the previous example:  

\[
\begin{bmatrix}
12 & 56 \\
45 & 1 \\
9 & 14
\end{bmatrix}
\]

Enter the matrix (or retrieve it from the previous example).  

Select the VAR menu and execute $MNX2$.  

Find the minimum element.
Applying a Program to an Array

APLY makes use of list processing to transform each element of an array according to a desired procedure. The input array must be numeric, but the output “array” may be symbolic. Since arrays cannot actually contain symbolic objects, a convention for symbolic “pseudo-arrays” is used. Each row of elements is grouped into a single list and the set of rows is grouped into a list. For example, a $2 \times 2$ pseudo-array looks like this:

$\begin{bmatrix}
\{ \text{element}_{11} & \text{element}_{12} \} \\
\{ \text{element}_{21} & \text{element}_{22} \}
\end{bmatrix}$

The procedure applied to each element must be a program that takes exactly one argument (i.e. the element) and returns exactly one result (i.e. the transformed element).

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>(\rightarrow)</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ array ]</td>
<td>&lt;&lt; program &gt;&gt;</td>
<td>(\rightarrow)</td>
<td>[[ array ]] or {{ array }}</td>
</tr>
</tbody>
</table>

Techniques used in APLY

- **Manipulating Meta-Objects.** *Meta-objects* are composite objects like arrays and lists that have been disassembled on the stack. APLY illustrates several approaches to manipulating the elements and dimensions of such objects.

- **Application of List Processing.** APLY makes use of DOSUBS (although DOLIST might also have been used) to perform the actual transformation of array elements.

- **Using an IFERR ... THEN ... ELSE ... END Structure.** The entire symbolic pseudo-array case is handled within a error structure—triggered when the \(\rightarrow\)ARRY command generates an error when symbolic elements are present.

- **Using Flags.** User flag 1 is used to track the case when the input array is a vector.
APLY program listing

Program:

\[
\begin{align*}
\rightarrow &\ a\ p \\
\rightarrow & \\
1 &\ CF \\
a &\ DUP\ SIZE \\
DUP\ SIZE \\
IF\ 1 &= \\
THEN\ 1\ SF\ 1\ + \\
SWAP\ OBJ\rightarrow\ OBJ\rightarrow\ DROP \\
1\ +\ ROLL \\
ELSE\ DROP2\ a\ OBJ\rightarrow \\
END\ DUP\ OBJ\rightarrow\ DROP\ * \\
SWAP\ OVER\ 2\ + \\
ROLLD\ \rightarrow\ LIST \\
1\ p\ DOSUBS
\end{align*}
\]

Comments:

Store the array and program in local variables.

Begin the main local variable structure.

Make sure the flag 1 is clear to begin the procedure.

Retrieve the dimensions of the array.

Determine if the array is a vector.

If array is a vector, set flag 1 and add a second dimension by treating the vector as an \( n \times 1 \) matrix.

Disassemble the original vector, leaving the element count, \( n \), in level 1.

Roll the elements up the stack and bring the “matrix” dimensions of the vector to level 1.

If array is a matrix, clean up the stack and decompose the matrix into its elements, leaving its dimension list on level 1.

Duplicate the dimension list and compute the total number of elements.

Roll up the element count and combine all elements into a list. Note that the elements in the list are in row-major order.

Recalls the program and uses it as an argument for DOSUBS (DOLIST works in this case as well). Result is a list of transformed elements.
Comments:
Disassembles the result list and brings the array dimensions to level 1.
Begins the error-trapping structure. Its purpose is to find and handle the cases when the result list contains symbolic elements.
Was original array a vector?
If the original array was a vector, then drop the second dimension (1) from the dimension list.
Convert the elements into a array with the given dimensions. If there are symbolic elements present, an error will be generated and the error-clause which follows will be executed.
Begin the error clause.
Put the array dimensions on levels 2 and 1. If the array is a vector, level 1 contains a 1.
Is original array a matrix? Clear flag 1 after performing the test.
Drop the number of matrix elements.
Store the array dimensions in local variables.
Begin local variable structure and initiate FOR..NEXT loop for each row.
Collect a group of elements into a row (a list).
Computes the number of elements to roll so that the next row can be collected.
Repeat loop for the next row.
Gather rows into a list, forming a list of lists (symbolic pseudo-array).

Program:
OBJ+ 1 + ROLL

IFERR

IF 1 FS?
THEN OBJ→ DROP →LIST

END →ARRAY

THEN
OBJ→

IF 1 FC?C
THEN DROP

END →n m

« 1 n
FOR i

m →LIST

' m+(n-i)+i ' EVAL
ROLLD

NEXT

n →LIST

Programming Examples 2-31
Program:

```plaintext
END 1 CF
```

Comments:
Close the local variable structure and end the IFERR..THEN..END structure. Clear flag 1 before exiting the program.

Stores the program in APLY.

Checksum: # 49768d
Bytes: 319

Example: Apply the function, \( f(x) = Ax^3 - 7 \) to each element \( x \) of the vector \([3 \ 2 \ 4 \ -8 \ 1]\).

Converts Between Number Bases

\( nBASE \) converts a positive decimal number \( (x) \) into a tagged string representation of the equivalent value in a different number base \( (b) \). Both \( x \) and \( b \) must be real numbers. \( nBASE \) automatically rounds both arguments to the nearest integer.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( b )</td>
<td>( \rightarrow )</td>
<td>( x ) base ( b ): &quot;string&quot;</td>
</tr>
</tbody>
</table>

2-32 Programming Examples
Techniques used in nBASE

- **String Concatenation and Character Manipulation.** nBASE makes use of several string and character manipulation techniques to build up the result string.

- **Tagged Output.** nBASE labels ("tags") the output string with its original arguments so that the output is a complete record of the command.

- **Indefinite Loops.** nBASE accomplishes most of its work using indefinite loops—both DO..UNTIL..END and WHILE..REPEAT..END loops.

### nBASE program listing

**Program:**

```plaintext
1 CF 0 RND SWAP 0 RND
⇒ b n

ª

n LOG b LOG /

10 RND

IP n 0

⇒ i m k
```

**Comments:**

- Clear flag 1 and round both arguments to integers.
- Store the base and number in local variables.
- Begin the outer local variable structure.
- Computes the ratio of the common logarithms of number and base.
- Rounds result to remove imprecision in last decimal place.
- Find the integer part of log ratio, recall the original number, and initialize the counter variable \( k \) for use in the DO..UNTIL loop.
- Store the values in local variables.
Program:
```
" 
"
DO

'm' EVAL b i
'k' EVAL - ^
DUP2 MOD

IF DUP 0 ==
 'm' EVAL b ≥
AND
THEN 1 SF
END 'm' STO
/ IP

IF DUP 10 ≥
THEN 55 + CHR

END +
'k' 1 STO+
```

Comments:
Begin inner local variable structure, enter an empty string and begin the DO..UNTIL..END loop.
Compute the decimal value of the \((i - k)\)th position in the string.
Makes a copy of the arguments and computes the decimal value still remaining that must be accounted for by other positions.
Is the remainder zero and \(m \geq b\)?
If the test is true, then set flag 1.
Store the remainder in \(m\).
Compute the number of times the current position-value goes into the remaining decimal value. This is the "digit" that belongs in the current position.
Is the "digit" \(\geq 10\)?
Then convert the digit into a alphabetic digit (such as A, B, C, \ldots).
Append the digit to the current result string and increment the counter variable \(k\).
Program:

```
UNTIL 'm' EVAL 0 ==

END
IF 1 FS?C
THEN 0 +
WHILE i 'k' EVAL
- 0 ≠
REPEAT 0 +
1 'k' STO+

END
END
```

```
" base" b +

n SWAP + →TAG

```

```
ENTER ' nBASE STO
```

Comments:
Repeat the DO..UNTIL loop until \( m = 0 \) (i.e. all decimal value has been accounted for).
Is flag 1 set? Clear the flag after the test.
Then add a placeholder zero to the result string.
Begin WHILE..REPEAT loop to determine if additional placeholder zeros are needed.
Loop repeats as long as \( i \neq k \).
Add an additional placeholder zero and increment \( k \) before repeating the test-clause.
End the WHILE..REPEAT..END loop, the IF..THEN..END structure, and the inner local variable structure.
End the outermost IF..THEN..ELSE..END structure and create the label string and tag the result string using the original arguments.

Stores the program in \( nBASE \).

Checksum: \# 36427d
Bytes: 416.5

Example: Convert 1000\(_{10}\) to base 23.

```
1000 ENTER 23 VAR nBASE
```

```
1: 1000 base23: "1KB"
```

Programming Examples 2-35
Verifying Program Arguments

The two utility programs in this section verify that the argument to a program is the correct object type.

- NAMES verifies that a list argument contains exactly two names.
- VFY verifies that the argument is either a name or a list containing exactly two names. It calls NAMES if the argument is a list.

You can modify these utilities to verify other object types and object content.

NAMES (Check List for Exactly Two Names)

If the argument for a program is a list (as determined by VFY), NAMES verifies that the list contains exactly two names. If the list does not contain exactly two names, an error message appears in the status area and program execution is aborted.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ valid list }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ invalid list }</td>
<td>→</td>
<td>(error message in status area)</td>
</tr>
</tbody>
</table>

Techniques used in NAMES

- Nested conditionals. The outer conditional verifies that there are two objects in the list. If so, the inner conditional verifies that both objects are names.
- Logical functions. NAMES uses the AND command in the inner conditional to determine if both objects are names, and the NOT command to display the error message if they are not both names.
Program:

```
IF
   OBJ>
   DUP 2 SAME
THEN
   DROP
   IF
      TYPE & SAME
   SWAP TYPE & SAME
   AND
   NOT
   THEN
      "List needs two names"
      DOERR
   END
ELSE
   DROPN
   "Illegal list size"
   DOERR
END
```

Comments:

 Starts the outer conditional structure.
 Returns the $n$ objects in the list to levels 2 through $(n + 1)$, and returns the list size $n$ to level 1.
 Copies the list size and tests if it's 2.
 If the size is 2, moves the objects to levels 1 and 2, and starts the inner conditional structure.
 Tests if the first object is a name: returns 1 if so, otherwise 0.
 Moves the second object to level 1, then tests if it is a name (returns 1 or 0).
 Combines test results: Returns 1 if both tests were true, otherwise returns 0.
 Reverses the final test result.
 If the objects are not both names, displays an error message and aborts execution.
 Ends the inner conditional structure.
 If the list size is not 2, drops the list size, displays an error message, and aborts execution.
 Ends the outer conditional.
 Stores the program in $\textit{NAMES}$. 
NAMES is demonstrated in the program VFY.

VFY (Verify Program Argument)

VFY verifies that an argument on the stack is either a name or a list that contains exactly two names.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td>'name'</td>
</tr>
<tr>
<td>{ valid list }</td>
<td>→</td>
<td>{ valid list }</td>
</tr>
<tr>
<td>{ invalid list}</td>
<td>→</td>
<td>{ invalid list} (and error message in status area)</td>
</tr>
<tr>
<td>invalid object</td>
<td>→</td>
<td>invalid object (and error message in status area)</td>
</tr>
</tbody>
</table>

Techniques used in VFY

- **Utility programs.** VFY by itself has little use. However, it can be used with minor modifications by other programs to verify that specific object types are valid arguments.

- **CASE ... END (case structure).** VFY uses a case structure to determine if the argument is a list or a name.

- **Structured programming.** If the argument is a list, VFY calls NAMES to verify that the list contains exactly two names.

- **Local variable structure.** VFY stores its argument in a local variable so that it can be passed to NAMES if necessary.

- **Logical function.** VFY uses NOT to display an error message.

Required Programs

- NAMES (page 2-36) verifies that a list argument contains exactly two names.
VFY program listing

Program:
«
DUP
DTAG
→ argm
«
CASE
argm TYPE 5 SAME
THEN
argm NAMES
END
argm TYPE 6 SAME NOT
THEN
"Not name or list"
DOERR
END
END
»
»
(ENTER) (VFY) STO

Comments:
Copies the original argument to leave on the stack.
Removes any tags from the argument for subsequent testing.
Stores the argument in local variable argm.
Begins the defining procedure.
Begins the case structure.
Tests if the argument is a list. If so, puts the argument back on the stack and calls NAMES to verify that the list is valid, then leaves the CASE structure.
Tests if the argument is not a name. If so, displays an error message and aborts execution.
Ends the CASE structure.
Ends the defining procedure.
Enters the program, then stores it in VFY.

Checksum: # 36796d
Bytes: 139.5

Example: Execute VFY to test the validity of the name argument BEN. (The argument is valid and is simply returned to the stack.)

\[1:\]
BEN (ENTER)
VAR \[VFY\]

Programming Examples 2-39
Example: Execute \textit{VFY} to test the validity of the list argument \{ \textit{BEN JEFF SARAH} \}. Use the name from the previous example, then enter the names \textit{JEFF} and \textit{SARAH} and convert the three names to a list.

\begin{verbatim}
JEFF (ENTER) SARAH (ENTER) 3 PRG LIST → LIST
\end{verbatim}

Execute \textit{VFY}. Since the list contains too many names, the error message is displayed and execution is aborted.

\begin{verbatim}
VAR \textit{VFY}
\end{verbatim}

Converting Procedures from Algebraic to RPN

This section contains a program, \( → \textit{RPN} \), that converts an algebraic expression into a series (list) of objects in equivalent RPN order. Note that \( → \textit{RPN} \) is a program provided with the \textit{TEACH} command. You can find it in the \textit{EXAMPLES} directory by pressing \texttt{EXAM} \texttt{PRGS} \texttt{→RPN}.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb'</td>
<td>→</td>
<td>{ objects }</td>
</tr>
</tbody>
</table>

Techniques used in \( → \textit{RPN} \)

- \textbf{Recursion}. The \( → \textit{RPN} \) program calls itself as a subroutine. This powerful technique works just like calling another subroutine as long as the stack contains the proper arguments before the program calls itself. In this case the level 1 argument is tested first to be sure that it is an algebraic expression before \( → \textit{RPN} \) is called again.
Object Type-Checking. →RPN uses conditional branching that depends on the object type of the level 1 object.

Nested Program Structures. →RPN nests IF ... THEN ... END structures inside FOR ... NEXT loops inside a IF ... THEN ... ELSE ... END structure.

List Concatenation. The result list of objects in RPN order is built by using the ability of the + command to sequentially append additional elements to a list. This is a handy technique for gathering results from a looping procedure.

→RPN program listing

Program:

```
OBJ
IF OVER
THEN → n f
```

Comments:

Take the expression apart. If the argument count is nonzero, then store the count and the function. Begins local variable defining procedure. Begins FOR ... NEXT loop, which converts any algebraic arguments to lists. Tests whether argument is an algebraic. If argument is an algebraic, convert it to a list first. Roll down the stack to prepare for the next argument. Repeat the loop for the next argument. Tests to see if level 1 object is a list. If not a list, then convert it to one. Ends the IF ... THEN ... END structure.
Program:

IF b

Tests to see if there is more
than one argument.

THEN 2 n

Combine all of the arguments
into a list.

START +

Append the function to the end
of the list.

NEXT

End the local variable defining
procedure.

END f +

For functions with no
arguments, converts to a simple
list.

End the IF ... THEN ...
ELSE ... END structure.

ELSE 1 \rightarrow LIST SWAP DROP

Example: Convert the following algebraic expression to a series of
objects in RPN syntax: 'A×COS(B+\sqrt(C/D))/3-X^3'.

Checksum: # 28598d
Bytes: 189.5

Checksum: # 28598d
Bytes: 189.5

Example: Convert the following algebraic expression to a series of
objects in RPN syntax: 'A×COS(B+\sqrt(C/D))/3-X^3'.

Checksum: # 28598d
Bytes: 189.5
Bessel Functions

This section contains a program, *BER*, that calculates the real part $\text{Ber}_n(x)$ of the Bessel function $J_n(xe^{3\pi i/4})$. When $n = 0$,

$$\text{Ber}(x) = 1 - \frac{(x/2)^4}{2!^2} + \frac{(x/2)^8}{4!^2} - \ldots$$

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>→</td>
<td>$\text{Ber}(z)$</td>
</tr>
</tbody>
</table>

Techniques used in *BER*

- **Local variable structure.** At its outer level, *BER* consists solely of a local variable structure and so has two properties of a user-defined function: it can take numeric or symbolic arguments from the stack, or it can take arguments in algebraic syntax. However, because *BER* uses a DO ... UNTIL ... END loop, its defining procedure is a program. (Loop structures are not allowed in algebraic expressions.) Therefore, unlike user-defined functions, *BER* is not differentiable.

- **DO ... UNTIL ... END loop (indefinite loop with counter).** *BER* calculates successive terms in the series using a counter variable. When the new term does not differ from the previous term to within the 12-digit precision of the calculator, the loop ends.

- **Nested local variable structures.** The outer structure is consistent with the requirements of a user-defined function. The inner structure allows storing and recalling of key parameters.
BER program listing

Program:

```
\( \times \)
\( \times/2 \) \( \Rightarrow \text{NUM} \ 2 \ 1 \)
\( \times \) \( \overline{2} \) \( j \) \( \text{sum} \)
```

Comments:

Creates local variable \( x \).
Begins outer defining procedure.
Enters \( x/2 \), the first counter value, and the first term of the series, then creates local variables.
Begins inner defining procedure.
Begins the loop.
Recalls the old sum and calculates the new sum.

```
\( \text{DO} \)
\( \text{sum} \)
\( \text{sum}+(-1)^{(j/2)} \times \overline{2} \times (2 \times j) / \text{SQ}(j!) \)
\( \text{EVAL} \)
\( 2 \ 'j' \text{STO}+ \)
\( \text{DUP 'sum' STO UNTIL} \)
\( == \)
\( \text{END sum} \)
\( \text{END} \)
```

Increments the counter.
Stores the new sum.
Ends the loop clause.
Tests the old and new sums.
Ends the loop.
Recalls the sum.
Ends inner defining procedure.
Ends outer defining procedure.

Stores the program in \( BER \).

Checksum: \# 36388d
Bytes: 200.5

Example: Calculate Ber(3).

```
\( \text{VAR} \)
3 \( \overline{BER} \)
```

Calculate Ber(2) in algebraic syntax.

```
\( \overline{BER} \) \( \text{2} \)
\( \text{EVAL} \)
```

2-44 Programming Examples
Animation of Successive Taylor’s Polynomials

This section contains three programs that manipulate graphics objects to display a sequence of Taylor’s polynomials for the sine function.

- **SINTP** draws a sine curve, and saves the plot in a variable.
- **SETTS** superimposes plots of successive Taylor’s polynomials on the sine curve plot from **SINTP**, and saves the resulting graphics objects in a list.
- **TSA** uses the ANIMATE command to display in succession each graphics object from the list built in **SETTS**.

SINTP (Converting a Plot to a Graphics Object)

**SINTP** draws a sine curve, returns the plot to the stack as a graphics object, and stores that graphics object in a variable. Make sure your calculator is in Radians mode.

Techniques used in SINTP

- **Programmatic use of PLOT commands.** **SINTP** uses PLOT commands to build and display a graphics object.
SINTP program listing

Program:
\[
\begin{align*}
&'\text{SIN}(x)' \text{ STEQ} \\
&\text{FUNCTION } '-2*\pi' \rightarrow \text{NUM} \\
&DUP \text{ NEG} \text{ XRNG} \\
&-2 \ 2 \ \text{XRNG} \\
&\text{ERASE} \text{ DRAW} \\
&\text{PICT} \text{ RCL} '\text{SINT}' \text{ STO} \\
&\end{align*}
\]

Comments:
Stores the expression for \( \sin x \) in \( \text{EQ} \).
Sets the plot type and \( x \)- and \( y \)-axis display ranges.
Erases \( \text{PICT} \), then plots the expression.
Recalls the resultant graphics object and stores it in \( \text{SINT} \).
Stores the program in \( \text{SINTP} \).

Checksum: \# 1971d
Bytes: 91.5

\( \text{SINTP} \) is demonstrated in the program \( \text{ TSA} \).

SETTS (Superimposing Taylor's Polynomials)

\( \text{SETTS} \) superimposes successive Taylor’s polynomials on a sine curve and stores each graphics object in a list.

Techniques used in SETTS

- **Structured programming.** \( \text{SETTS} \) calls \( \text{SINTP} \) to build a sine curve and convert it to a graphics object.

- **FOR ... STEP (definite loop).** \( \text{SETTS} \) calculates successive Taylor’s polynomials for the sine function in a definite loop. The loop counter serves as the value of the order of each polynomial.

- **Programmatic use of PLOT commands.** \( \text{SETTS} \) draws a plot of each Taylor’s polynomial.

- **Manipulation of graphics objects.** \( \text{SETTS} \) converts each Taylor’s polynomial plot into a graphics object. Then it executes \( + \) to combine each graphics object with the sine curve stored in \( \text{SINT} \), creating nine new graphics objects, each the superposition of a...
Taylor’s polynomial on a sine curve. SETTS then puts the nine new graphics objects, and the sine curve graphics object itself, in a list.

**SETTS program listing**

**Program:**

```plaintext
\( \langle \ 
\) SETTP

1 17 FOR n

'SIN(X)' 'X' n TAYLR
STEP ERASE DRAW
PICT RCL SINT +

2 STEP

SINT
10 \( \rightarrow \) LIST
'TSL' STO

\( \rangle \)
```

**Comments:**

Plots a sine curve and stores the graphics object in SINT.
Sets the range for the FOR loop using local variable \( n \).
Plots the Taylor’s polynomial of order \( n \).
Returns the plot to the stack as a graphics object and executes + to superimpose the sine plot from SINT.
Increments the loop counter \( n \) by 2 and repeats the loop.
Puts the sine curve graphics object on the stack, then builds a list containing it and the nine graphics objects created in the loop. Stores the list in TSL.

Stores the program in SETTS.

Checksum: \( \# \ 28102d \)
Bytes: \( 138.5 \)

SETTS is demonstrated in the program TSA.

**TSA (Animating Taylor’s Polynomials)**

TSA displays in succession each graphics object created in SETTS.

**Techniques used in TSA**

- **Passing a global variable.** Because SETTS takes several minutes to execute, TSA does not call SETTS. Instead, you must first execute SETTS to create the global variable TSL containing the list of
graphics objects. TSA simply executes that global variable to put the list on the stack.

FOR . . . NEXT (definite loop). TSA executes a definite loop to display in succession each graphics object from the list.

**TSA program listing**

**Program:**

```
«
TSL OBJ+ 
( ( #0 #0 ) .5 0 ) + ANIMATE 11 DROPN 
»
```

**Comments:**

- Puts the list TSL on the stack and converts it to 10 graphics objects and the list count.
- Set up the parameters for ANIMATE.
- Displays the graphics in succession.
- Removes the graphics objects and list count from the stack.
- Stores the program in TSA.

Checksum: # 59350d

Bytes: 96.5

**Example:** Execute SETTS and TSA to build and display in succession a series of Taylor’s polynomial approximations of the sine function.

Set Radians mode and execute SETTS to build the list of graphics objects. (SETTS takes several minutes to execute.) Then execute TSA to display each plot in succession. The display shows TSA in progress.

RAD (if necessary)

SETTS TSA
Press (CANCEL) to stop the animation. Press (⇒) RAD to restore Degrees mode.

---

**Programmatic Use of Statistics and Plotting**

This section describes a program *PIE* you can use to draw pie charts. *PIE* prompts for single variable data, stores that data in the statistics matrix $\Sigma$DAT, then draws a labeled pie chart that shows each data point as a percentage of the total.

**Techniques used in PIE**

- **Programmatic use of PLOT commands.** *PIE* executes XRNG and YRNG to define $x$- and $y$-axis display ranges in user units, and executes ARC and LINE to draw the circle and individual slices.

- **Programmatic use of matrices and statistics commands.**

- **Manipulating graphics objects.** *PIE* recalls PICT to the stack and executes GOR to merge the label for each slice with the plot.

- **FOR ... NEXT (definite loop).** Each slice is calculated, drawn, and labeled in a definite loop.

- **CASE ... END structure.** To avoid overwriting the circle, each label is offset from the midpoint of the arc of the slice. The offset for each label depends on the position of the slice in the circle. The CASE ... END structure assigns an offset to the label based on the position of the slice.

- **Preserving calculator flag status.** Before specifying Radians mode, *PIE* saves the current flag status in a local variable, then restores that status at the end of the program.

- **Nested local variable structures.** At different parts of the process, intermediate results are saved in local variables for convenient recall as needed.

- **Temporary menu for data input.**
PIE program listing

Program:
```
«
RCLF \flags
«
RAD
( "SLICE" Σ+ )
( )
( "CLEAR" CLZ )
( ) ( )
( "DRAW" CONT )
»
TMENU
"Key values into SLICE; DRAW restarts program."
PROMPT
ERASE 1 131 XRNGL
erase 1 64 YRNGLCLLC
"Please wait...«
Drawing Pie Chart"
1 DISP
(66,32) 20 0 6.28
ARC
PICT RCL →LCD
RCLZ TOT /
DUP 100 *
→prcnts
«
2 π →NUM * *
0
»
Comments:
Recalls the current flag status and stores it in variable flags.

Sets Radians mode.
Defines the input menu: Key 1 executes Σ+ to store each data point in ΣDAT, key 3 clears ΣDAT, and key 6 continues program execution after data entry.
Displays the temporary menu.
Prompts for inputs.
# represents the newline character (↕) after you enter the program on the stack.
Erases the current PICT and sets plot parameters.
Displays "drawing" message.

Draws the circle.

Displays the empty circle.
Recalls the statistics data matrix, computes totals, and calculates the proportions.
Converts the proportions to percentages.
Stores the percentage matrix in prcnts.

Multiplies the proportion matrix by 2π, and enters the initial angle (0).
Program:

\[ \text{\( \ast prop\ angle \)} \]

\[
\begin{align*}
\text{\( \ast prop\ SIZE\ OBJ\)} & \\
\text{\( \ast DROP\ SWAP\)} & \\
\text{\( \ast FOR\ n\)} & \\
\text{\( \ast (66,32)\ prop\ n\ GET\)} & \\
\text{\( \ast 'angle'\ STO+\)} & \\
\text{\( \ast \)} & \\
\text{\( \ast angle\ COS\ angle\ SIN\)} & \\
\text{\( \ast R+C\ 20\ \ast\ OVER\ +\)} & \\
\text{\( \ast LINE\)} & \\
\text{\( \ast PICT\ RCL\)} & \\
\text{\( \ast angle\ prop\ n\ GET\)} & \\
\text{\( \ast 2\ /\ -\ DUP\ DUP\)} & \\
\text{\( \ast \cos\ SWAP\ SIN\ R+C\)} & \\
\text{\( \ast 26\ \ast\ (66,32)\ +\)} & \\
\text{\( \ast SWAP\)} & \\
\text{\( \ast CASE\)} & \\
\text{\( \ast DUP\ 1.5\ \leq\ )\}} & \\
\text{\( \ast THEN\)} & \\
\text{\( \ast DROP\)} & \\
\text{\( \ast END\)} & \\
\text{\( \ast DUP\ 4.4\ \leq\ )\}} & \\
\text{\( \ast THEN\)} & \\
\text{\( \ast DROP\ 15\ -\ )\}} & \\
\text{\( \ast END\)} & \\
\text{\( \ast 5\ <\ )\}} & \\
\text{\( \ast THEN\)} & \\
\text{\( \ast (3,2)\ +\ )\}} & \\
\text{\( \ast END\)} & \\
\text{\( \ast END\)} & \\
\end{align*}
\]

Comments:

Stores the angle matrix in \textit{prop} and angle in \textit{angle}.

Sets up 1 to \textit{m} as loop counter range.

Begins loop-clause.

Puts the center of the circle on the stack, then gets the \(n\)th value from the proportion matrix and adds it to \textit{angle}.

Computes the endpoint and draws the line for the \(n\)th slice.

Recalls \textit{PICT} to the stack.

For labeling the slice, computes the midpoint of the arc of the slice.

Starts the CASE structure to test \textit{angle} and determine the offset value for the label.

From 0 to 1.5 radians, doesn’t offset the label.

From 1.5 to 4.4 radians, offsets the label 15 user units left.

From 4.4 to 5 radians, offsets the label 3 units right and 2 units up.

Ends the CASE structure.
Comments:

Gets the $n$th value from the percentage matrix, rounds it to one decimal place, and converts it to a string with “%” appended.

Converts the string to a graphics object.

Adds the label to the plot and stores the new plot.

Displays the updated plot.

Ends the loop structure.

Displays the finished plot.

Restores the original flag status.

Restores the previous menu.

(You must first press **CANCEL** to clear the plot.)

Stores the program in *PIE*.

---

**Example:** The inventory at Fruit of the Vroom, a drive-in fruit stand, includes 983 oranges, 416 apples, and 85 bananas. Draw a pie chart to show each fruit’s percentage of total inventory.
Clear the current statistics data. (The prompt is removed from the display.) Key in the new data and draw the pie chart.

```
CLEAR
983 SLICE
416 SLICE
85 SLICE
DRAW
```

Press [CANCEL] to return to the stack display.

---

**Trace Mode**

This section contains two programs, $\alpha$ENTER and $\beta$ENTER, which together provide "trace mode" for the HP 48 using an external printer. To turn on "trace mode," set flag $-63$ and activate User mode. To turn off "trace mode," clear flag $-63$ or turn off User mode.

**Techniques used in $\alpha$ENTER and $\beta$ENTER**

- **Vectored ENTER.** Setting flag $-63$ and activating User mode turns on vectored ENTER. When vectored ENTER is turned on and variable $\alpha$ENTER exists, the command-line text is put on the stack as a string and $\alpha$ENTER is evaluated. Then, if variable $\beta$ENTER exists, the command that triggered the command-line processing is put on the stack as a string and $\beta$ENTER is evaluated.

**$\alpha$ENTER program listing**

```
\begin{verbatim}
\alpha \text{ENTER}
\end{verbatim}
```

**Comments:**

- Prints the command line text, then converts the string to an object and evaluates it.

```
\begin{verbatim}
\alpha \text{ENTER} \text{ STO}
\end{verbatim}
```

Stores the program in $\alpha$ENTER. (Press $\alpha$ A to type $\alpha$. You must use this name.)
\textbf{\boldmath $\beta$ENTER program listing}

\begin{verbatim}
Program:
\( \langle \)
\texttt{PR1 DROP PRSTC}
\( \rangle \)
\texttt{ENTER \, \betaENTER \, STO}
\end{verbatim}

Comments:
Prints the command that caused the processing, then drops it and prints the stack in compact form.

Stores the program in $\beta$ENTER. (Press $\alpha \rightarrow B$ to type $\beta$. You must use this name.)

Checksum: # 37631d
Bytes: 28

---

\textbf{Inverse-Function Solver}

This section describes the program \textit{ROOTR}, which finds the value of $x$ at which $f(x) = y$. You supply the variable name for the program that calculates $f(x)$, the value of $y$, and a guess for $x$ (in case there are multiple solutions).

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>\rightarrow</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'{\textit{function name}'}</td>
<td>$y$</td>
<td>$x_{\text{guess}}$</td>
<td>$\rightarrow$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

\textbf{Techniques used in \textit{ROOTR}}

- **Programmatic use of root-finder.** \textit{ROOTR} executes ROOT to find the desired $x$-value.

- **Programs as arguments.** Although programs are commonly named and then executed by calling their names, programs can also be put on the stack and used as arguments to other programs.

2-54 Programming Examples
ROOTR program listing

Program:
```
«
  + fname yvalue xguess  «
  xguess 'XTEMP' STO
```
```
  XTEMP fname
  yvalue - »
  'XTEMP'
  xguess
  ROOT
»
'XTEMP' PURGE
»
```

Comments:
- Creates local variables.
- Begins the defining procedure.
- Creates variable \( XTEMP \) (to be solved for).
- Enters program that evaluates \( f(x) - y \).
- Enters name of unknown variable.
- Enters guess for \( XTEMP \).
- Solves program for \( XTEMP \).
- Ends the defining procedure.
- Purges the temporary variable.
- Stores the program in \( ROOTR \).

Checksum: \# 13007d
Bytes: 163

Example: Assume you often work with the expression \( 3.7x^3 + 4.5x^2 + 3.9x + 5 \) and have created the program \( X\rightarrow FX \) to calculate the value:
```
«
  + x '3.7*x^3+4.5*x^2+3.9*x+5'
»
```
You can use \( ROOTR \) to calculate the inverse function.

Example: Find the value of \( x \) for which \( X\rightarrow FX \) equals 599.5. Use a guess in the vicinity of 1.

Start by keying in \( X\rightarrow FX \):
```
3: 1: «
  + x '3.7*x^3+4.5*
  x^2+3.9*x+5'
»
```

Programming Examples 2-55
Store the program in $X \rightarrow FX$, then enter the program name, the $y$-value 599.5, and the guess 1, and execute $ROOTR$:

```
\begin{array}{l}
\text{\texttt{1 X \rightarrow FX STO}} \\
\text{\texttt{1 VAR X + FX ENTER}} \\
\text{\texttt{599.5 ENTER 1 ROOTR}} \\
\end{array}
```

---

**Animating a Graphical Image**

Program *WALK* shows a small person walking across the display. It animates this custom graphical image by incrementing the image position in a loop structure.

**Techniques used in WALK**

- **Custom graphical image.** (Note that the programmer compiles the full information content of the graphical image before writing the program by building the image *interactively* in the Graphics environment and then returning it to the command line.)

- **FOR . . . STEP (definite loop).** *WALK* uses this loop to animate the graphical image. The ending value for the loop is $MAXR$. Since the counter value cannot exceed $MAXR$, the loop executes indefinitely.
WALK program listing

Program:

\[
\begin{align*}
\text{GROB} & \ 9 \ 15 \ E300 \\
140015001C001400E300 \\
8000C110AA0094009000 \\
4100220014102800 \\
\end{align*}
\]

\begin{align*}
+ \text{walk} & \\
\end{align*}

\begin{align*}
\text{ERASE} \ ( \ # \ 0d \ # \ 0d \ ) & \\
\text{PYVIEW} & \\
( \ # \ 0d \ # \ 25d \ ) & \\
\text{PICT} & \ \text{OVER} \ \text{walk} \ \text{GXOR} \\
5 \ \text{MAXR} & \ \text{FOR} \ i \\
\end{align*}

\begin{align*}
i & \ 131 \ \text{MOD} \ \Rightarrow \ B \\
\# & \ 25d \ 2 \ \Rightarrow \text{LIST} \\
\text{PICT} & \ \text{OVER} \ \text{walk} \ \text{GXOR} \\
\text{PICT} & \ \text{ROT} \ \text{walk} \ \text{GXOR} \\
5 \ \text{STEP} & \\
\end{align*}

Comments:

- Puts the graphical image of the walker in the command line. (Note that the hexadecimal portion of the graphics object is a continuous integer \(E300 \ldots 2800\). The linebreaks do not represent spaces.) Creates local variable \textit{walk} containing the graphics object.
- Clears \textit{PICT}, then displays it.
- Puts the first position on the stack and turns on the first image. This readies the stack and \textit{PICT} for the loop.
- Starts the loop to generate horizontal coordinates indefinitely.
- Computes the horizontal coordinate for the next image.
- Specifies a fixed vertical coordinate. Puts the two coordinates in a list.
- Displays the new image, leaving its coordinates on the stack.
- Turns off the old image, removing its coordinates from the stack.
- Increments the horizontal coordinate by 5.
- Stores the program in \textit{WALK}.

Checksum: \# 18146d
Bytes: 240.5
Example: Send the small person out for a walk.

VAR WALK

Press CANCEL when you think the walker's tired.
This chapter contains an alphabetical listing of the programmable commands and functions available on the HP 48. The listings include the following information:

- a brief definition of what the command or function does
- a stack diagram showing the arguments it requires (if any)
- the keys to press to gain access to it
- any flags that may affect how it works
- additional information about how it works and how to use it
- an example of its use
- related commands or functions

The next few pages explain how to read the stack diagrams in the command reference, how commands are alphabetized, and the meaning of the command classifications at the upper right corner of each stack diagram.

How to Read Stack Diagrams

Each entry in the command reference includes a stack diagram. This is a table showing the arguments that the command, function, or analytic function takes from the stack and the results that it returns to the stack. The “→” character in the table separates the arguments from the results. The stack diagram for a command may contain more than one “argument → result” line, reflecting all possible combinations of arguments and results for that command.
Consider this example:

**ACOS**

**Arc Cosine Analytic Function:** Returns the value of the angle having the given cosine.

This diagram indicates that the *analytic function* ACOS (*Arc Cosine*) takes a single argument from level 1 and returns one result (to level 1). ACOS can take either a real or complex number or an algebraic object as its argument. In the first case, it returns the numeric arccosine; in the second, it returns the symbolic arccosine expression of the argument.

Some commands affect a calculator state—a mode, a reserved variable, a flag, or a display— without taking any arguments from the stack or returning any results to the stack. No stack diagrams are shown for these commands.

**Parallel Processing with Lists**

Commands that can use the parallel list processing feature are denoted by the "{}" symbol located above the stack diagram. This feature is discussed in greater detail in Appendix G.

As a rule-of-thumb, a command can use parallel list processing if all the following are true:

- The command checks for valid argument types. Commands that apply to all object types, such as DUP, SWAP, ROT, and so forth, do not use parallel list processing.
- The command takes exactly one, two, three, four, or five arguments, none of which may itself be a list. Commands, such as →LIST,
that have an indefinite number of arguments do not use parallel list processing.

- The command isn't a programming branch command (IF, FOR, CASE, NEXT, and so forth).

The HP 48 also has a few commands (PURGE and DELKEYS are examples) that have list processing capability built into their definitions, and so do not also use the parallel list processing feature.

**How Commands Are Alphabetized**

Commands appear in alphabetical order. Command names that contain special (non-alphabetic) characters are organized as follows:

- For commands that contain both special and alphabetic characters:
  - A special character at the start of a command name is ignored. Therefore, the command *H follows the command GXOR and precedes the command HALT.
  - A special character *within* or at the *end* of a command name is considered to follow "Z" at the end of the alphabet. Therefore, the command R—B follows the command RSD and precedes the command R—C.

- Commands that contain *only* special characters appear at the end of the dictionary.

**Classification of Operations**

The command dictionary contains HP 48 *commands, functions, and analytic functions*. Commands are calculator operations that can be executed from a program. Functions are commands that can be included in algebraic objects. Analytic functions are functions for which the HP 48 provides an inverse and a derivative. There are also four non-programmable *operations* (DBG, NEXT, SST, and SST↓) that are included with the programmable commands as a convenience because they are used interactively while programming.

The definitions of the abbreviations used for argument and result objects are contained in the following table, "Terms Used in Stack Diagrams." Often, descriptive subscripts are added to convey more information.
## Terms Used in Stack Diagrams

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg</td>
<td>Argument</td>
</tr>
<tr>
<td>[ array ]</td>
<td>Real or complex vector or matrix.</td>
</tr>
<tr>
<td>[ C-array ]</td>
<td>Complex vector or matrix.</td>
</tr>
<tr>
<td>date</td>
<td>Date in form MM.DDYYYY or DD.MMYYYY.</td>
</tr>
<tr>
<td>{ dim }</td>
<td>List of one or two array dimensions (real numbers).</td>
</tr>
<tr>
<td>'global'</td>
<td>Global name.</td>
</tr>
<tr>
<td>grob</td>
<td>Graphics object.</td>
</tr>
<tr>
<td>HMS</td>
<td>A real-number time or angle in hours-minutes-seconds format.</td>
</tr>
<tr>
<td>{ list }</td>
<td>List of objects.</td>
</tr>
<tr>
<td>local</td>
<td>Local name.</td>
</tr>
<tr>
<td>[[ matrix ]]</td>
<td>Real or complex matrix.</td>
</tr>
<tr>
<td>n or m</td>
<td>Positive integer real number (rounded if noninteger).</td>
</tr>
<tr>
<td>:n_port:namebackup</td>
<td>Backup identifier.</td>
</tr>
<tr>
<td>:n_port:n.library</td>
<td>Library identifier.</td>
</tr>
<tr>
<td>#n</td>
<td>Binary integer.</td>
</tr>
<tr>
<td>{ #n #m }</td>
<td>Pixel coordinates. (Uses binary integers.)</td>
</tr>
<tr>
<td>'name'</td>
<td>Global or local name.</td>
</tr>
<tr>
<td>obj</td>
<td>Any object.</td>
</tr>
<tr>
<td>PICT</td>
<td>Current graphics object.</td>
</tr>
<tr>
<td>$&lt;$ program $&gt;$</td>
<td>Program.</td>
</tr>
<tr>
<td>[ R-array ]</td>
<td>Real vector or matrix.</td>
</tr>
<tr>
<td>&quot;string&quot;</td>
<td>Character string.</td>
</tr>
<tr>
<td>'symb'</td>
<td>Expression, equation, or name treated as an algebraic.</td>
</tr>
<tr>
<td>T/F</td>
<td>Test result used as an argument: zero (false) or non-zero (true) real number.</td>
</tr>
<tr>
<td>0/1</td>
<td>Test result returned by a command: zero (false) or one (true).</td>
</tr>
<tr>
<td>time</td>
<td>Time in form HH.MMSSs.</td>
</tr>
<tr>
<td>[ vector ]</td>
<td>Real or complex vector.</td>
</tr>
<tr>
<td>x or y</td>
<td>Real number.</td>
</tr>
<tr>
<td>x_unit</td>
<td>Unit object, or a real number treated as a dimensionless object.</td>
</tr>
<tr>
<td>(x,y)</td>
<td>Complex number in rectangular form, or user-unit coordinate.</td>
</tr>
<tr>
<td>z</td>
<td>Real or complex number.</td>
</tr>
</tbody>
</table>
ABS

Absolute Value Function: Returns the absolute value of its argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>→</td>
<td>$</td>
</tr>
<tr>
<td>$(x,y)$</td>
<td>→</td>
<td>$\sqrt{x^2 + y^2}$</td>
</tr>
<tr>
<td>$x_unit$</td>
<td>→</td>
<td>$</td>
</tr>
<tr>
<td>[ array ]</td>
<td>→</td>
<td>$|array|$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'ABS(symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access:
- [MTH] REAL [NXT] ABS
- [MTH] MATR NORM ABS
- [MTH] NXT CMPL ABS
- [MTH] VECTR ABS

Affected by Flags: Numerical Results (−3)

Remarks: ABS has a derivative (SIGN) but not an inverse.

In the case of an array, ABS returns the Frobenius (Euclidean) norm of the array, defined as the square root of the sum of the squares of the absolute values of all $n$ elements. That is,

$$\sqrt{\sum_{i=1}^{n} |z_i|^2}$$

Related Commands: NEG, SIGN
**ACK**

**Acknowledge Alarm Command:** Acknowledges the oldest past-due alarm.

**Keyboard Access:** \( \text{TIME} \) ALRM \( \text{ACK} \)

**Affected by Flags:** Repeat Alarms Not Rescheduled (–43), Acknowledged Alarms Saved (–44)

**Remarks:** ACK clears the alert annunciator if there are both no other past-due alarms and no other active alert sources (such as a low battery condition).

ACK has no effect on control alarms. Control alarms that come due are automatically acknowledged and saved in the system alarm list.

**Related Commands:** ACKALL

---

**ACKALL**

**Acknowledge All Alarms Command:** Acknowledges all past-due alarms.

**Keyboard Access:** \( \text{TIME} \) ALRM \( \text{ACKA} \)

**Affected by Flags:** Repeat Alarms Not Rescheduled (–43), Acknowledged Alarms Saved (–44)

**Remarks:** ACKALL clears the alert annunciator if there are no other active alert sources (such as a low battery condition).

ACKALL has no effect on control alarms. Control alarms that come due are automatically acknowledged and saved in the system alarm list.

**Related Commands:** ACK
**Arc Cosine Analytic Function:** Returns the value of the angle having the given cosine.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z )</td>
<td>→</td>
<td>( \arccos z )</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'ACOS(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(<-)(ACOS)\)

**Affected by Flags:** Principal Solution \((-1)\), Numerical Results \((-3)\), Angle Mode \((-17, -18)\)

**Remarks:** For a real argument \( x \) in the domain \(-1 \leq x \leq 1\), the result ranges from 0 to 180 degrees (0 to \( \pi \) radians; 0 to 200 grads).

A real argument outside of this domain is converted to a complex argument \( z = x + 0i \), and the result is complex.

The inverse of \( \cos \) is a *relation*, not a function, since \( \cos \) sends more than one argument to the same result. The inverse relation for \( \cos \) is expressed by ISOL as the *general solution*

\[ '\pm 1*\cos(Z) + 2*\pi*n' \]

The function \( \cos \) is the inverse of a *part* of \( \cos \), a part defined by restricting the domain of \( \cos \) such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of \( \cos \) are called the *principal values* of the inverse relation. \( \cos \) in its entirety is called the *principal branch* of the inverse relation, and the points sent by \( \cos \) to the boundary of the restricted domain of \( \cos \) form the *branch cuts* of \( \cos \).

The principal branch used by the HP 48 for \( \cos \) was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.
The graphs below show the domain and range of ACOS. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation \( z \neq 1 \times \text{ACOS}(Z) + 2\pi n I \) for the case \( s I = 1 \) and \( n I = 0 \). For other values of \( s I \) and \( n I \), the vertical band in the lower graph is translated to the right or to the left. Taken together, the bands cover the whole complex plane, which is the domain of COS.

View these graphs with domain and range reversed to see how the domain of COS is restricted to make an inverse function possible. Consider the vertical band in the lower graph as the restricted domain \( Z = (x, y) \). COS sends this domain onto the whole complex plane in the range \( W = (u, v) = \text{COS}(x, y) \) in the upper graph.

**Related Commands:** ASIN, ATAN, COS, ISOL
Inverse Hyperbolic Cosine Analytic Function: Returns the inverse hyperbolic cosine of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z )</td>
<td>→</td>
<td>( \text{acosh}\ z )</td>
</tr>
<tr>
<td>( '\text{symb}' )</td>
<td>→</td>
<td>( '\text{ACOSH}(\text{symb})' )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{MTH HYP ACOSH} \)

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

Remarks: For real arguments \( |x| < 1 \), ACOSH returns the complex result obtained for the argument \( (x, 0) \).

The inverse of ACOSH is a relation, not a function, since COSH sends more than one argument to the same result. The inverse relation for COSH is expressed by ISOL as the general solution

\[ '\pm1\times\text{ACOSH}(z)+2\times\pi\times i\times n1' \]

The function ACOSH is the inverse of a part of COSH, a part defined by restricting the domain of COSH such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of COSH are called the principal values of the inverse relation. ACOSH in its entirety is called the principal branch of the inverse relation, and the points sent by ACOSH to the boundary of the restricted domain of COSH form the branch cuts of ACOSH.

The principal branch used by the HP 48 for ACOSH was chosen because it is analytic in the regions where the arguments of the real-valued inverse function are defined. The branch cut for the complex-valued hyperbolic arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ACOSH. The graph of the domain shows where the branch cut occurs: the heavy solid line...
ACOSH

marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation \( s1*ACOSH(Z) + 2*\pi*i*n1 \) for the case \( s1=1 \) and \( n1=0 \). For other values of \( s1 \) and \( n1 \), the horizontal half-band in the lower graph is rotated to the left and translated up and down. Taken together, the bands cover the whole complex plane, which is the domain of COSH.

View these graphs with domain and range reversed to see how the domain of COSH is restricted to make an inverse function possible. Consider the horizontal half-band in the lower graph as the restricted domain \( Z = (x, y) \). COSH sends this domain onto the whole complex plane in the range \( W = (u, v) = COSH(x, y) \) in the upper graph.

**Related Commands:** ASINH, ATANH, COSH, ISOL
ADD

Add List Command: Adds corresponding elements of two lists or adds a number to each of the elements of a list.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ list}_1 }</td>
<td>{ list}_2 }</td>
<td>→</td>
<td>{ list}_result }</td>
</tr>
<tr>
<td>{ list }</td>
<td>obj_{non-list}</td>
<td>→</td>
<td>{ list}_result }</td>
</tr>
<tr>
<td>obj_{non-list}</td>
<td>{ list }</td>
<td>→</td>
<td>{ list}_result }</td>
</tr>
</tbody>
</table>

Keyboard Access: [MTH] LIST ADD

Affected by Flags: None

Remarks: ADD executes the + command once for each of the elements in the list. If two lists are the arguments, they must have the same number of elements as ADD will execute the + command once for each corresponding pair of elements. If one argument is a non-list object, ADD will attempt to execute the + command using the non-list object and each element of the list argument, returning the result to the corresponding position in the result. (See the + command entry to see the object combinations that are defined.) If an undefined addition is encountered, a Bad Argument Type error results.

Related Commands: ΔLIST, ΠLIST, ΣLIST
**ALOG**

**Common Antilogarithm Analytic Function:** Returns the common antilogarithm; that is, 10 raised to the given power.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z )</td>
<td>→</td>
<td>( 10^z )</td>
</tr>
<tr>
<td>('symb')</td>
<td>→</td>
<td>('ALOG(symb)')</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(10^z\)

**Affected by Flags:** Numerical Results \((-3)\)

**Remarks:** For complex arguments:

\[
10^{(x,y)} = e^{cx} \cos cy + i e^{cx} \sin cy
\]

where \( c = \ln 10 \).

**Related Commands:** EXP, LN, LOG

---

**AMORT**

**Amortize Command:** Amortizes a loan or investment based upon the current amortization settings.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>→</td>
<td>principal</td>
<td>interest</td>
<td>balance</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{SOLVE} \ AMORT\)

**Affected by Flags:** Financial Payment Mode \((-14)\)
Remarks: Values must be stored in the TVM variables ($I\%YR$, $PV$, $PMT$, and $PYR$). The number of payments $n$ is taken from level 1 and flag $-14$.

Related Commands: TVM, TVMBEG, TVMEND, TVMROOT

AND

And Function: Returns the logical AND of two arguments.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$#n_1$</td>
<td>$#n_2$</td>
<td>→</td>
<td>$#n_3$</td>
</tr>
<tr>
<td>&quot;string_1&quot;</td>
<td>&quot;string_2&quot;</td>
<td>→</td>
<td>&quot;string_3&quot;</td>
</tr>
<tr>
<td>$T/F_1$</td>
<td>$T/F_2$</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>$T/F$</td>
<td>'symb'</td>
<td>→</td>
<td>'T/F AND symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$T/F$</td>
<td>→</td>
<td>'symb AND T/F'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>'symb_2'</td>
<td>→</td>
<td>'symb_1 AND symb_2'</td>
</tr>
</tbody>
</table>

Keyboard Access:

MTH BASE NXT LOGIC AND

PRG TEST NXT AND

Affected by Flags: Numerical Results ($-3$), Binary Integer Wordsize ($-5$ through $-10$)

Remarks: When the arguments are binary integers or strings, AND does a bit-by-bit (base 2) logical comparison.

- An argument that is a binary integer is treated as a sequence of bits as long as the current wordsize. Each bit in the result is determined by comparing the corresponding bits ($bit_1$ and $bit_2$) in the two arguments as shown in the following table:
AND

<table>
<thead>
<tr>
<th>$bit_1$</th>
<th>$bit_2$</th>
<th>$bit_1$ AND $bit_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

An argument that is a string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must have the same number of characters.

When the arguments are real numbers or symbolics, AND simply does a true/false test. The result is 1 (true) if both arguments are non-zero; it is 0 (false) if either or both arguments are zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic expressions, then the result is an algebraic of the form '$symb_1$ AND $symb_2$'. Execute $\rightarrow$NUM (or set flag –3 before executing AND) to produce a numeric result from the algebraic result.

**Related Commands:** NOT, OR, XOR

---

ANIMATE

**Animate Command:** Displays graphic objects in sequence.

<table>
<thead>
<tr>
<th>Level n+1...Level 2</th>
<th>Level 1</th>
<th>(\rightarrow)</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$grob_n \ldots grob_1$</td>
<td>$n_{\text{grobs}}$</td>
<td>$\rightarrow$ same stack</td>
<td></td>
</tr>
<tr>
<td>$grob_n \ldots grob_1$</td>
<td>{ $n$ { $#X$ $#Y$ } delay rep }</td>
<td>$\rightarrow$ same stack</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:**

| PRG | GROB | NXT | ANIM |

**Affected by Flags:** None

3-14 Command Reference
**Remarks:** ANIMATE displays a series of graphics objects (or variables containing them) one after the other. You can use a list to specify the area of the screen you want to animate (pixel coordinates \(#X\) and \(#Y\)), the number of seconds before the next grob is displayed (delay), and the number of times the sequence is run (rep). If rep is set to 0, the sequence is played one million times, or until you press [CANCEL].

If you use a list on level 1, all parameters must be present.

The animation displays PICT while displaying the grobs. The grobs and the animate parameters are left on the stack.

**Example:** The following program draws half a cylinder and rotates it:

```plaintext
« PARSURFACE { 'COS(X)' 'SIN(X)' Y } STEQ « I 180 I + XXRNG ERASE DRAW PICT RCL » I 0 359 8 SEQ OBJ→ ANIMATE DROPN »
```

This program also illustrates the use of SEQ and PARSURFACE. You can adjust the increment value used with SEQ (8 is used here) to change the number of images drawn by the program, or to use less memory.

---

**APPLY**

**Apply to Arguments Function:** Creates an expression from the specified function name and arguments.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1 →</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ symb₁ \ldots symbₙ }</td>
<td>'name'</td>
<td>'name(symb₁ \ldots symbₙ)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\leftarrow\) [SYMBOLIC] [NXT] APPLY
**APPLY**

**Affected by Flags:** None

**Remarks:** A user-defined function \( f \) that checks its arguments for special cases often can’t determine whether a symbolic argument \( x \) represents one of the special cases. The function \( f \) can use APPLY to create a new expression \( \text{\textquoteleft} f(x) \text{\textquoteright} \). If the user now evaluates \( \text{\textquoteleft} f(x) \text{\textquoteright} \), \( x \) is evaluated before \( f \), so the argument to \( f \) will be the result obtained by evaluating \( x \).

The algebraic syntax for APPLY is this:

\[
\text{\textquoteleft}\text{\textsc{apply}}(\text{\textit{name}}, \text{\textit{symb}_1}, \ldots, \text{\textit{symb}_n})\text{\textquoteright}
\]

When evaluated in an algebraic expression, APPLY evaluates the arguments (to resolve local names in user-defined functions) before creating the new object.

**Example:** The following user-defined function \( \text{Asin} \) is a variant of the built-in function \( \text{ASIN} \). \( \text{Asin} \) checks for special numerical arguments. If the argument on the stack is symbolic (the second case in the case structure), \( \text{Asin} \) uses APPLY to return the expression \( \text{\textquoteleft}\text{Asin}(\text{argument})\text{\textquoteright} \).

```plaintext
\% \% argument
\% CASE
-9 FS? THEN argument ASIN END
(6 7 9) argument TYPE POS
THEN \text{\textquoteleft}\text{\textsc{apply}}(\text{\textit{Asin}}, \text{\textit{argument}})\text{\textquoteright} EVAL END
\text{\textquoteleft}argument==1\text{\textquoteright} THEN \text{\textquoteleft}\pi/2\text{\textquoteright} END
\text{\textquoteleft}argument==0\text{\textquoteright} THEN \text{\textquoteleft}-\pi/2\text{\textquoteright} END
\% \%
END
\% \%
<ENTER> 0 Asin <STO>
```

**Related Commands:** QUOTE, |
**ARC**

**Draw Arc Command:** Draws an arc in *PICT* counterclockwise from $x_{\theta_1}$ to $x_{\theta_2}$, with its center at the coordinate specified in level 4 and its radius specified in level 3.

<table>
<thead>
<tr>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x, y)$</td>
<td>$x_{\text{radius}}$</td>
<td>$x_{\theta_1}$</td>
<td>$x_{\theta_2}$</td>
<td>→</td>
<td>$x_{\theta_2}$</td>
</tr>
<tr>
<td>$#n$ $#m$</td>
<td>$#n_{\text{radius}}$</td>
<td>$x_{\theta_1}$</td>
<td>$x_{\theta_2}$</td>
<td>→</td>
<td>$x_{\theta_2}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (PRG) FILT HEL

**Affected by Flags:** Angle Mode (−17 and −18)

The setting of flags −17 and −18 determine the interpretation of $x_{\theta_1}$ and $x_{\theta_2}$ (degrees, radians, or grads).

**Remarks:** ARC always draws an arc of constant radius in pixels, even when the radius and center are specified in user-units, regardless of the relative scales in user-units of the $x$- and $y$-axes. With user-unit arguments, the arc starts at the pixel specified by $(x, y) + (a, b)$, where $(a, b)$ is the rectangular conversion of the polar coordinate $(x_{\text{radius}}, x_{\theta_1})$. The resultant distance in pixels from the starting point to the center pixel is used as the actual radius, $r'$. The arc stops at the pixel specified by $(r', x_{\theta_2})$.

If $x_{\theta_1} = x_{\theta_2}$, ARC plots one point. If $|x_{\theta_1} - x_{\theta_2}| > 360$ degrees, $2\pi$ radians, or 400 grads, ARC draws a complete circle.

**Example:** In Degrees mode, with the $x$-axis display range (XRNG) specified as −6.5 to 6.5, the command sequence $\langle 0, 0 \rangle 1 \ 0 \ 90 \ ARC$ draws an arc counterclockwise from 0 to 90 degrees with a constant radius of 10 pixels.

**Related Commands:** BOX, LINE, TLINE
**ARCHIVE**

**Archive HOME Command:** Creates a backup copy of the *HOME* directory (that is, all variables), the user-key assignments, and the alarm catalog in the specified backup object (\( : n_{port} : name \)) in independent RAM.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( :n_{port} :name )</td>
<td>( \rightarrow )</td>
<td>( \rightarrow )</td>
</tr>
<tr>
<td>( :IO :name )</td>
<td>( \rightarrow )</td>
<td>( \rightarrow )</td>
</tr>
</tbody>
</table>

**Keyboard Access:**  
\[ \text{←MEMORY} \text{ NXT ARCHI} \]

**Affected by Flags:** I/O Device (−33), I/O Messages (−39) if the argument is \( : IO : name \)

**Remarks:** The specified port number can be 0 through 33. The port used (except 0) must be configured as independent RAM. (See FREE.) An error will result if there is not enough independent RAM in the specified port to copy the HOME directory.

If the backup object is \( : IO : name \), then the copied directory is transmitted in binary via Kermit protocol through the current I/O port to the specified filename.

To save flag settings, execute RCLF and store the resulting list in a variable.

**Related Commands:** RESTORE
ARG

Argument Function: Returns the (real) polar angle $\theta$ of a complex number $(x, y)$.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x, y)$</td>
<td>$\rightarrow$</td>
<td>$\theta$</td>
</tr>
<tr>
<td>&quot;symb&quot;</td>
<td>$\rightarrow$</td>
<td>'ARG(symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access: [MTH] [NXT] [CMPL] [ARG]

Affected by Flags: Angle mode $(-17, -18)$

Remarks: The polar angle $\theta$ is equal to:
- $\arctan y/x$ for $x \geq 0$
- $\arctan y/x + \pi \text{ sign } y$ for $x < 0$, Radians mode
- $\arctan y/x + 180 \text{ sign } y$ for $x < 0$, Degrees mode
- $\arctan y/x + 200 \text{ sign } y$ for $x < 0$, Grads mode

A real argument $x$ is treated as the complex argument $(x, 0)$.

ARRY →

Array to Stack Command: Takes an array and returns its elements as separate real or complex numbers. Also returns a list of the dimensions of the array.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level $nm+1 \ldots$ Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ vector ]</td>
<td>$\rightarrow$</td>
<td>$z_1 \ldots z_n$</td>
<td>${ n_{element} }$</td>
</tr>
<tr>
<td>[[ matrix ]]</td>
<td>$\rightarrow$</td>
<td>$z_{11} \ldots z_{nm}$</td>
<td>${ n_{row} ; m_{col} }$</td>
</tr>
</tbody>
</table>
ARY→

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: The command OBJ→ includes this functionality. ARRY→ is included for compatibility with the HP 28S. ARRY→ is not in a menu.

If the argument is an n-element vector, the first element is returned to level n + 1 (not level nm + 1), and the nth element to level 2.

Related Commands: →ARY, DTAG, EQ→, LIST→, OBJ→, STR→

→ARY

Stack to Array Command: Returns a vector of n real or complex elements or a matrix of n × m real or complex elements.

<table>
<thead>
<tr>
<th>Level nm+1 ... Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>z₁ ... zₙ</td>
<td>n_element</td>
<td>→</td>
<td>[ vector ]</td>
</tr>
<tr>
<td>z₁₁ ... zₙₘ</td>
<td>{ nᵣow mᵦol }</td>
<td>→</td>
<td>[ [ matrix ] ]</td>
</tr>
</tbody>
</table>

Keyboard Access: PRG TYPE ARRY

Affected by Flags: None

Remarks: The elements of the result array should be entered into the stack in row order, with z₁₁ (or z₁) in level nm + 1 (or n + 1), and zₙₘ (or zₙ) in level 2. If one or more of the elements is a complex number, the result array will be complex.

Related Commands: ARRY→, LIST→, →LIST, OBJ→, STR→, →TAG, →UNIT
**Arc Sine Analytic Function:** Returns the value of the angle having the given sine.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>→</td>
<td>$\text{arc sin } z$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'ASIN(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (&)(ASIN)

**Affected by Flags:** Principal Solution (−1), Numerical Results (−3), Angle Mode (−17, −18)

**Remarks:** For a real argument $x$ in the domain $−1 \leq x \leq 1$, the result ranges from $−90$ to $+90$ degrees ($−\pi/2$ to $+\pi/2$ radians; $−100$ to $+100$ grads).

A real argument outside of this domain is converted to a complex argument $z = x + 0i$, and the result is complex.

The inverse of SIN is a relation, not a function, since SIN sends more than one argument to the same result. The inverse relation for SIN is expressed by ISOL as the general solution

\[ '\text{ASIN}(Z)\equiv(-1)^n1+\pi\times n1' \]

The function ASIN is the inverse of a part of SIN, a part defined by restricting the domain of SIN such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of SIN are called the principal values of the inverse relation. ASIN in its entirety is called the principal branch of the inverse relation, and the points sent by ASIN to the boundary of the restricted domain of SIN form the branch cuts of ASIN.

The principal branch used by the HP 48 for ASIN was chosen because it is analytic in the regions where the arguments of the real-valued inverse function are defined. The branch cut for the complex-valued arc sine function occurs where the corresponding real-valued function...
ASIN

is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ASIN. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation \( \text{ASIN}(Z^{\ast}(n) + \pi \ast n) \) for the case \( n1 = 0 \). For other values of \( n1 \), the vertical band in the lower graph is translated to the right (for \( n1 \) positive) or to the left (for \( n1 \) negative). Taken together, the bands cover the whole complex plane, which is the domain of SIN.

View these graphs with domain and range reversed to see how the domain of SIN is restricted to make an inverse function possible. Consider the vertical band in the lower graph as the restricted domain \( Z = (x, y) \). SIN sends this domain onto the whole complex plane in the range \( W = (u, v) = \text{SIN}(x, y) \) in the upper graph.

**Related Commands:** ACOS, ATAN, ISOL, SIN
ASINH

Arc Hyperbolic Sine Analytic Function: Returns the inverse hyperbolic sine of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>→</td>
<td>asinh z</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'ASINH(symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access: \[ \text{MTH} \quad \text{HYP} \quad \text{ASINH} \]

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

Remarks: The inverse of \( \text{SINH} \) is a relation, not a function, since \( \text{SINH} \) sends more than one argument to the same result. The inverse relation for \( \text{SINH} \) is expressed by ISOL as the general solution

\[ '\text{ASINH}(z)*(-1)^n*\pi*i*n1' \]

The function \( \text{ASINH} \) is the inverse of a part of \( \text{SINH} \), a part defined by restricting the domain of \( \text{SINH} \) such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of \( \text{SINH} \) are called the principal values of the inverse relation. \( \text{ASINH} \) in its entirety is called the principal branch of the inverse relation, and the points sent by \( \text{ASINH} \) to the boundary of the restricted domain of \( \text{SINH} \) form the branch cuts of \( \text{ASINH} \).

The principal branch used by the HP 48 for \( \text{ASINH} \) was chosen because it is analytic in the regions where the arguments of the real-valued function are defined. The branch cut for the complex-valued \( \text{ASINH} \) function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graph for \( \text{ASINH} \) can be found from the graph for \( \text{ASIN} \) (see \( \text{ASIN} \)) and the relationship \( \text{asinh } z = -i \text{ asin } iz \).

Related Commands: \( \text{ACOSH}, \text{ATANH}, \text{ISOL}, \text{SINH} \)
ASN

Assign Command: Defines a single key on the user keyboard by assigning the given object to the key \( x_{\text{key}} \), which is specified as \( rc.p \).

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{obj} )</td>
<td>( x_{\text{key}} )</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>'SKEY'</td>
<td>( x_{\text{key}} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{MODES} \text{ KEYS: ASN} \)

Affected by Flags: User-Mode Lock \((-61)\) and User Mode \((-62)\) affect the status of the user keyboard

Remarks: The argument \( x_{\text{key}} \) is a real number \( rc.p \) specifying the key by its row number \( r \), column number \( c \), and plane (shift) \( p \). The legal values for \( p \) are as follows:

<table>
<thead>
<tr>
<th>Plane, ( p )</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or 1</td>
<td>unshifted</td>
</tr>
<tr>
<td>2</td>
<td>( \leftarrow ) left-shifted</td>
</tr>
<tr>
<td>3</td>
<td>( \rightarrow ) right-shifted</td>
</tr>
<tr>
<td>4</td>
<td>( @ ) alpha-shifted</td>
</tr>
<tr>
<td>5</td>
<td>( @ \leftarrow ) alpha left-shifted</td>
</tr>
<tr>
<td>6</td>
<td>( @ \rightarrow ) alpha right-shifted</td>
</tr>
</tbody>
</table>

Once ASN has been executed, pressing a given key in User or 1-User mode executes the user-assigned object. The user key assignment remains in effect until the assignment is altered by ASN, STOKEYS, or DELKEYS. Keys without user assignments maintain their standard definitions.

If the argument \( \text{obj} \) is the name 'SKEY', then the specified key is restored to its standard key assignment on the user keyboard. This
is meaningful only when all standard key assignments had been suppressed (for the user keyboard) by the command 'S' DELKEYS (see DELKEYS).

To make multiple key assignments simultaneously, use STOKEYS. To delete key assignments, use DELKEYS.

Be careful not to reassign or suppress the keys necessary to cancel User mode. If this happens, exit User mode by doing a system halt ("warm start"): press and hold (ON) and the C key simultaneously, releasing the C key first. This cancels User mode.

Example: Executing ASN with GETI in level 2 and 85.3 in level 1 assigns GETI to (.Navigate) ("7) on the user keyboard. (Navigate) ("7) has a location of 85.3 because it is eight rows down, five columns across, and right-shifted.) When the calculator is in User mode, pressing (Navigate) ("7) now executes GETI (instead of executing ("7)).

Related Commands: DELKEYS, RCLKEYS, STOKEYS

---

**ASR**

**Arithmetic Shift Right Command:** Shifts a binary integer one bit to the right, except for the most significant bit, which is maintained.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH BASE NXT BIT ASR

**Affected by Flags:** Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

**Remarks:** The most significant bit is preserved while the remaining \( \text{wordsize} - 1 \) bits are shifted right one bit. The second-most significant bit is replaced with a zero. The least significant bit is shifted out and lost.

---

Command Reference 3-25
An arithmetic shift is useful for preserving the sign bit of a binary integer that will be shifted. Although the HP 48 makes no special provision for signed binary integers, you can still interpret a number as a signed quantity.

**Related Commands:** SL, SLB, SR, SRB

---

**ATAN**

**Arc Tangent Analytic Function:** Returns the value of the angle having the given tangent.

\[
\begin{array}{c|c}
\text{Level 1} & \rightarrow \\
\hline
z & \text{arc tan } z \\
'symb' & '\text{ATAN}(symb)' \\
\end{array}
\]

**Keyboard Access:** \(\text{ATAN}\)

**Affected by Flags:** Principal Solution \((-1)\), Numerical Results \((-3)\), Angle Mode \((-17, -18)\)

**Remarks:** For a real argument, the result ranges from \(-90\) to \(+90\) degrees \((-\pi/2\) to \(+\pi/2\) radians; \(-100\) to \(+100\) grads).

The inverse of TAN is a relation, not a function, since TAN sends more than one argument to the same result. The inverse relation for TAN is expressed by ISOL as the general solution

\[{}'\text{ATAN}(z)+\pi*n1'\]

The function ATAN is the inverse of a part of TAN, a part defined by restricting the domain of TAN such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of TAN are called the principal values of the inverse relation. ATAN in its entirety is called the principal branch of the inverse relation, and the points sent by ATAN to the boundary of the restricted domain of TAN form the branch cuts of ATAN.
The principal branch used by the HP 48 for ATAN was chosen because it is analytic in the regions where the arguments of the real-valued inverse function are defined. The branch cuts for the complex-valued arc tangent function occur where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ATAN. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation \( \text{ATAN}(Z) + \pi n \) for the case \( n = 0 \). For other values of \( n \), the vertical band in the lower graph is translated to the right (for \( n > 0 \)) or to the left (for \( n < 0 \)). Taken together, the bands cover the whole complex plane, which is the domain of TAN.

View these graphs with domain and range reversed to see how the domain of TAN is restricted to make an inverse function possible. Consider the vertical band in the lower graph as the restricted domain \( Z = \langle x, y \rangle \). TAN sends this domain onto the whole complex plane in the range \( W = \langle u, v \rangle = \text{TAN}(x, y) \) in the upper graph.

**Related Commands:** ACOS, ASIN, ISOL, TAN
ATAN

Domain: \( Z = (x, y) \)

Range: \( W = (u, v) = \text{ATAN}(x, y) \)

Branch Cuts for ATAN(Z)

ATANH

Arc Hyperbolic Tangent Analytic Function: Returns the inverse hyperbolic tangent of the argument.

\[
\begin{array}{c|c}
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
z & \rightarrow & \text{atanh } z \\
'symb' & \rightarrow & '\text{ATANH}(symb)' \\
\end{array}
\]

Keyboard Access: \( \text{MTH} \) \( \text{HYP} \) \( \text{ATAN} \)

Affected by Flags: Principal Solution \((-1)\), Numerical Results \((-3)\), Infinite Result Exception \((-22)\)

3-28 Command Reference
Remarks: For real arguments \(|x| > 1\), ATANH returns the complex result obtained for the argument \((x, 0)\). For a real argument \(x=\pm 1\), an Infinite Result exception occurs. If flag \(-22\) is set (no error), the sign of the result (MAXR) matches that of the argument.

The inverse of TANH is a relation, not a function, since TANH sends more than one argument to the same result. The inverse relation for TANH is expressed by ISOL as the general solution

\[
\text{ATANH}(z) + \pi i n
\]

The function ATANH is the inverse of a part of TANH, a part defined by restricting the domain of TANH such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of TANH are called the principal values of the inverse relation. ATANH in its entirety is called the principal branch of the inverse relation, and the points sent by ATANH to the boundary of the restricted domain of TANH form the branch cuts of ATANH.

The principal branch used by the HP 48 for ATANH was chosen because it is analytic in the regions where the arguments of the real-valued function are defined. The branch cut for the complex-valued ATANH function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graph for ATANH can be found from the graph for ATAN (see ATAN) and the relationship \(\text{atanh } z = -i \text{ atan } iz\).

Related Commands: ACOSH, ASINH, ISOL, TANH
ATICK

Axes Tick-Mark Command: Sets the axes tick-mark annotation in the reserved variable PPAR.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>#n</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ x,y }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ #n #m}</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: 🔄 PLOT PPAR NEXT ATICK

Affected by Flags: None

Remarks: Given x, ATICK sets the tick-mark annotation to x units on both the x- and the y-axis. For example, 2 would place tick-marks every 2 units on both axes.

Given #n, ATICK sets the tick-mark annotation to #n pixels on both the x- and the y-axis. For example, #5 would place tick-marks every 5 pixels on both axes.

Given { x,y }, ATICK sets the tick-mark unit annotation for each axis individually. For example, { 10 3 } would mark the x-axis at every multiple of 10 units, and the y-axis at every multiple of 3 units.

Given { #n #m } ATICK sets the tick-mark pixel annotation for each axis individually. For example, { 6 2 } would mark the x-axis every 6 pixels, and the y-axis every 2 pixels.

Related Commands: AXES, DRAX
**ATTACH**

**Attach Library Command:** Attaches the library with the specified number to the current directory. Each library has a unique number. If a port number is specified, it is ignored.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{\text{library}} )</td>
<td>→</td>
<td>( n_{\text{library}} )</td>
</tr>
<tr>
<td>( \text{:port} ; n_{\text{library}} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{LIBRARY} \) \( \text{NXT} \) \text{ATTACH}

**Affected by Flags:** None

**Remarks:** To use a library object, it must be in a port and it must be attached. A library object from an application card (ROM) is automatically in a port (1-33), but a library object copied into RAM (such as through the PC Link) must be stored into a port using STO.

Many libraries are attached automatically when an application card is installed. Others require you to ATTACH them, as do many libraries copied into RAM. (The owner’s manual for the application card or library will tell you which of its library objects must be attached manually.) You can also ascertain whether a library is attached to the current directory by executing LIBS.

A library that has been copied into RAM and then stored (with STO) into a port can be attached only after the calculator has been turned off and then on again following the STO command. This action (off/on) creates a system halt, which makes the library object “attachable.” Note that it also clears the stack, local variables, and the LAST stack, and it displays the MATH menu. (To save the stack first, execute \text{DEPTH \#LIST 'name' STO}.)

The number of libraries that can be attached to the HOME directory is limited only by the available memory. However, only one library at a time can be attached to any other directory. If you attempt to attach a second library to a non-HOME directory, the new library will overwrite the old one.
ATTACH

Related Commands: DETACH, LIBS

AUTO

Autoscale Command: Calculates a $y$-axis display range, or an $x$- and $y$-axis display range.

Keyboard Access: ←PLOT NXT AUTO

Affected by Flags: None

Remarks: The action of AUTO depends on the plot type as follows:

<table>
<thead>
<tr>
<th>Plot Type</th>
<th>Scaling Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION</td>
<td>Samples the equation in $EQ$ at 40 values of the independent variable, equally spaced through the $x$-axis plotting range, discards points that return $\pm\infty$, then sets the $y$-axis display range to include the maximum, minimum, and origin.</td>
</tr>
<tr>
<td>CONIC</td>
<td>Sets the $y$-axis scale equal to the $x$-axis scale.</td>
</tr>
<tr>
<td>POLAR</td>
<td>Samples the equation in $EQ$ at 40 values of the independent variable, equally spaced through plotting range, discards points that return $\pm\infty$, then sets both the $x$- and $y$-axis display ranges in the same manner as for plot type FUNCTION.</td>
</tr>
<tr>
<td>PARAMETRIC</td>
<td>Same as POLAR.</td>
</tr>
<tr>
<td>TRUTH</td>
<td>No action.</td>
</tr>
<tr>
<td>BAR</td>
<td>Sets the $x$-axis display range from 0 to the number of elements in $\Sigma DAT$, plus 1. Sets the $y$-range to the minimum and maximum of the elements. The $x$-axis is always included.</td>
</tr>
<tr>
<td>Plot Type</td>
<td>Scaling Action</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HISTOGRAM</td>
<td>Sets the x-axis display range to the minimum and maximum of the elements in $ΣDAT$. Sets the y-axis display range from 0 to the number of rows in $ΣDAT$.</td>
</tr>
<tr>
<td>SCATTER</td>
<td>Sets the x-axis display range to the minimum and maximum of the independent variable column (XCOL) in $ΣDAT$. Sets the y-axis display range to the minimum and maximum of the dependent variable column (YCOL).</td>
</tr>
</tbody>
</table>

AUTO does not affect 3D plots.

AUTO actually calculates a y-axis display range and then expands that range so that the menu labels do not obscure the resultant plot.

AUTO does not draw a plot—execute DRAW to do so.

**Example:** The program « FUNCTION AUTO DRAW DRA X » sets the plot type to FUNCTION, autoscales the y-axis, plots the equation in $EQ$, and adds axes to the plot.

**Related Commands:** DRAW, *H, SCALE, SCLΣ, *W, XRNG, YRNG

**AXES**

**Axes Command:** Specifies the intersection coordinates of the x- and y-axes, tick-mark annotation, and the labels for the x- and y-axes. This information is stored in the reserved variable $PPAR$.
**AXES**

**Keyboard Access:**  

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>← PLOT PPAR NXT AXES</td>
<td></td>
</tr>
</tbody>
</table>

**Affected by Flags:** None

**Remarks:** The argument for AXES (a complex number or list) is stored as the fifth parameter in the reserved variable `PPAR`. How the argument is used depends on the type of object it is:

- If the argument is a complex number, it replaces the current entry in `PPAR`.
- If the argument is a list containing any or all of the above variables, only variables that are specified are affected.

`atick` has the same format as the argument for the ATICK command. This is the variable that is affected by the ATICK command.

The default value for AXES is `(0, 0)`.

Axes labels are not displayed in `PICT` until subsequent execution of `LABEL`.

**Example:** The command sequence

```
( (0,0) 2 "t" "y" ) AXES LABEL
```

specifies an axes intersection at `(0,0)`, tick-mark annotation every 2 units, and puts the labels `t` and `y` in `PICT`. The labels are positioned to identify the horizontal and vertical axes respectively.

**Related Commands:** ATICK, DRAW, DRAX, LABEL

---

**BAR**

**Bar Plot Type Command:** Sets the plot type to BAR.

**Keyboard Access:**  

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>← PLOT NXT STAT PTYPE BAR</td>
<td></td>
</tr>
</tbody>
</table>

**Affected by Flags:** None

**Remarks:** When the plot type is BAR, the DRAW command plots a bar chart using data from one column of the current statistics matrix (reserved variable `ΣDAT`). The column to be plotted is specified by the XCOL command, and is stored in the first parameter of the reserved variable `ΣPAR`. The plotting parameters are specified in the reserved variable `PPAR`, which has the following form:
BAR

\{(x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}}) \, \text{indep} \, \text{res} \, \text{axes} \, \text{ptype} \, \text{depend}\ \}\}

For plot type BAR, the elements of \( PPAR \) are used as follows:

- \((x_{\text{min}}, y_{\text{min}})\) is a complex number specifying the lower left corner of \( PICT \) (the lower left corner of the display range). The default value is \((-6.5, -3.1)\).

- \((x_{\text{max}}, y_{\text{max}})\) is a complex number specifying the upper right corner of \( PICT \) (the upper right corner of the display range). The default value is \((6.5, 3.2)\).

- \text{indep} is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers, with the smaller of the numbers specifying the horizontal location of the first bar. The default value of \text{indep} is \( X \).

- \text{res} is a real number specifying the bar width in user-unit coordinates, or a binary integer specifying the bar width in pixels. The default value is \( 0 \), which specifies a bar width of 1 in user-unit coordinates.

- \text{axes} is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is \((0, 0)\).

- \text{ptype} is a command name specifying the plot type. Executing the command BAR places the command name BAR in \( PPAR \).

- \text{depend} is a name specifying a label for the vertical axis. The default value is \( Y \).

A bar is drawn for each element of the column in \( \Sigma DAT \). Its width is specified by \text{res} and its height is the value of the element. The location of the first bar can be specified by \text{indep}; otherwise, the value in \((x_{\text{min}}, y_{\text{min}})\) is used.

Related Commands: CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE
**BARPLOT**

**Draw Bar Plot Command:** Plots a bar chart of the specified column of the current statistics matrix (reserved variable $\Sigma DAT$).

**Keyboard Access:** $\left[\text{STAT}\right]$ PLOT BARPLOT

**Affected by Flags:** None

**Remarks:** The data column to be plotted is specified by XCOL and is stored as the first parameter in reserved variable $\Sigma PAR$. The default column is 1. Data can be positive or negative, resulting in bars above or below the axis. The $y$-axis is autoscaled, and the plot type is set to BAR.

When BARPLOT is executed from a program, the resulting plot does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

**Related Commands:** FREEZE, HISTPLOT, PICTURE, PVIEW, SCATRPLOT, XCOL

---

**BAUD**

**Baud Rate Command:** Specifies bit-transfer rate.

\[
\begin{array}{cc}
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
n_{\text{baudrate}} & \rightarrow & \\
\end{array}
\]

**Keyboard Access:** $\left[\text{I/O}\right]$ IOPAR BAUD

**Affected by Flags:** None

**Remarks:** Legal baud rates are 1200, 2400, 4800, and 9600 (default).

For more information, refer also to the reserved variable IOPAR ($I/O$ parameters) in appendix D, “Reserved Variables.”

**Related Commands:** CKSM, PARITY, TRANSIO

3-36  Command Reference
**BESTFIT**

**BEEP**

**Beep Command:** Sounds a tone at $n$ hertz for $x$ seconds.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{frequency}}$</td>
<td>$x_{\text{duration}}$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] [NXT] [OUT] [NXT] [BEEP]

**Affected by Flags:** Error Beep (–56)

**Remarks:** The frequency of the tone is subject to the resolution of the built-in tone generator. The maximum frequency is approximately 4400 Hz; the maximum duration is 1048.575 seconds. Arguments greater than these maximum values default to the maxima.

**Related Commands:** HALT, INPUT, PROMPT, WAIT

---

**BESTFIT**

**Best-Fitting Model Command:** Executes LR with each of the four curve fitting models, and selects the model yielding the largest correlation coefficient.

**Keyboard Access:** [STAT] [PAR] [MODL] [BESTF]

**Affected by Flags:** None

**Remarks:** The selected model is stored as the fifth parameter in the reserved variable $\Sigma PAR$, and the associated regression coefficients, intercept and slope, are stored as the third and fourth parameters, respectively. For a description of $\Sigma PAR$, see appendix D, “Reserved Variables.”

**Related Commands:** EXPFIT, LINFIT, LOGFIT, LR, PWRFIT
BIN

Binary Mode Command: Selects binary base for binary integer operations. (The default base is decimal.)

Keyboard Access: [MTH] [BASE] [BIN]

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: Binary integers require the prefix #. Binary integers entered and returned in binary base automatically show the suffix b. If the current base is not binary, binary numbers can still be entered by using the suffix b (the numbers are displayed in the current base, however).

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Related Commands: DEC, HEX, OCT, STWS, RCWS

BINS

Sort Into Frequency Bins Command: Sorts the elements of the independent column (XCOL) of the current statistics matrix (the reserved variable ΣDAT) into (n_bins + 2) bins, where the left edge of bin 1 starts at value x_min and each bin has width x_width.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_min</td>
<td>x_width</td>
<td>n_bins</td>
<td>[n_{bin1} ... n_{bin_n}]</td>
<td>[n_{bin_L} n_{bin_R}]</td>
</tr>
</tbody>
</table>

Keyboard Access: [STAT] [VAR] [BINS]

Affected by Flags: None

Remarks: BINS returns a matrix containing the frequency of occurrences in each bin, and a 2-element array containing the frequency of occurrences falling below or above the defined range of x-values. The array can be stored into the reserved variable ΣDAT.
and used to plot a bar histogram of the bin data (for example, by executing BARPLOT).

For each element \( x \) in \( YDAT \), the \( n \)th bin count \( n_{freq\ bin\ n} \) is incremented, where:

\[
n_{freq\ bin\ n} = IP \left[ \frac{x - x_{\min}}{x_{\text{width}}} \right]
\]

for \( x_{\min} \leq x \leq x_{\max} \), where \( x_{\max} = x_{\min} + (n_{\text{bins}})(x_{\text{width}}) \).

**Example:** If the independent column of \( YDAT \) contains the following data:

\[
7 2 3 1 4 6 9 0 1 1 3 5 1 3 2 6 9 5 8 5
1 2 5
\]

1 2 5 BINS returns \([5 3 5 2 2]\) and \([1 1]\).

The data has been sorted into 5 bins of width 2, starting at \( x \)-value 1 and ending at \( x \)-value 11. The first element of the matrix shows that 5 \( x \)-values fell in bin 1, where bin 1 ranges from \( x \)-value 1 through 2.99999999999. The vector shows that one \( x \)-value was less than \( x_{\min} \) (9), and one was greater than \( x_{\max} \) (13).

**Related Commands:** BARPLOT, XCOL

---

**BLANK**

**Blank Graphics Object Command:** Creates a blank graphics object of the specified width and height.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n_{width}</td>
<td>#m_{height}</td>
<td>→</td>
<td>grob_{blank}</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] [GROB] [BLANK]

**Affected by Flags:** None

**Related Commands:** →GROB, LCD→
Box Command: Draws in PICT a box whose opposite corners are defined by the specified pixel or user-unit coordinates.

\[
\begin{array}{c|c|c}
\text{Level 2} & \text{Level 1} & \rightarrow \\
\{ #n_1 \ #m_1 \} & \{ #n_2 \ #m_2 \} & \rightarrow \\
(x_1, y_1) & (x_2, y_2) & \rightarrow \\
\end{array}
\]

Keyboard Access: \texttt{PRG PICT BOX}

Affected by Flags: None

Related Commands: ARC, LINE, TLINE

BufLEN

Buffer Length Command: Returns the number of characters in the HP 48’s serial input buffer and a single digit indicating whether an error occurred during data reception.

\[
\begin{array}{c|c|c}
\text{Level 1} & \rightarrow & \text{Level 2} & \text{Level 1} \\
\rightarrow & n_{\text{chars}} & 0/1
\end{array}
\]

Keyboard Access: \texttt{I/O \textcircled{NXT} \texttt{SERIA BufLE}}

Affected by Flags: None

Remarks: The digit returned to level 1 is 1 if no framing, UART overrun, or input-buffer overflow errors occurred during reception, or 0 if one of these errors did occur. (The input buffer holds up to 255 bytes.) When a framing or overrun error occurs, data reception ceases until the error is cleared (which BufLEN does); therefore, \(n\) represents the data received before the error.
Use ERRM to see which error has occurred when BUFLEN returns 0 to level 1.

Related Commands: CLOSEIO, OPENIO, SBRK, SRECV, STIME, XMIT

BYTES

Byte Size Command: Returns the number of bytes and the checksum for the given object.

---

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>#n_{\text{checksum}}</td>
<td>\text{x}_{\text{size}}</td>
</tr>
</tbody>
</table>

Keyboard Access: \(\text{MEMORY}\)\(\text{BYTES}\)

Affected by Flags: None

Remarks: If the argument is a built-in object, then the size is 2.5 bytes and the checksum is \# 0.

If the argument is a global name, then the size represents the name and its contents, while the checksum represents the contents only. The size of the name alone is \((3.5 + 2 \times n)\), where \(n\) is the number of characters in the name.

Example: Objects that decompile identically can have different byte sizes and checksums. For instance,

\[
\{1\}
\]

and

\[
1 \ 'A' \ \text{STO} \ A \ \{\} +
\]

both produce lists containing the number 1. However, the first list contains the built-in object 1 (for a size of 7.5 bytes), while the second list contains a RAM copy of 1 (for a size of 15.5 bytes).

Related Commands: MEM
**B→R**

**Binary to Real Command:** Converts a binary integer to its floating-point equivalent.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n</td>
<td>→</td>
<td>n</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH BASE B→R

**Affected by Flags:** Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

**Remarks:** If \#n ≥ #1000000000000 (base 10), only the 12 most significant decimal digits are preserved in the resulting mantissa.

**Related Commands:** R→B

---

**CASE**

**CASE Conditional Structure Command:** Starts CASE ... END conditional structure.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>THEN</td>
<td>→</td>
<td>T/F</td>
</tr>
<tr>
<td>END</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG BRCH CASE CASE

**Affected by Flags:** None

---

3-42 Command Reference
Remarks: The CASE ... END structure executes a series of cases (tests). The first test that returns a true result causes execution of the corresponding true-clause, ending the CASE ... END structure. A default clause can also be included: this clause executes if all tests evaluate to false.

The CASE ... END structure has this syntax:

```
CASE
  test-clause1 THEN true-clause1 END
  test-clause2 THEN true-clause2 END
  ...
  test-clauseN THEN true-clauseN END

END
```

When CASE executes, test-clause1 is evaluated. If the test is true, true-clause1 executes, then execution skips to END. If test-clause1 is false, test-clause2 executes. Execution within the CASE structure continues until a true clause is executed, or until all the test clauses evaluate to false. If the default clause is included, it executes if all test clauses evaluate to false.

Example: The following program takes a numeric argument from the stack:

- if the argument is negative, it is added to itself
- if the argument is positive, it is negated
- if the argument is zero, the program aborts

```
<< CASE
  'x>0'
  THEN x NEG END
  'x<0'
  THEN x DUP + END
  'x==0'
  THEN 0 DOERR END
END
```
CASE

Related Commands: END, IF, IFERR, THEN

CEIL

Ceiling Function: Returns the smallest integer greater than or equal to the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>→</td>
<td>( n )</td>
</tr>
<tr>
<td>( x\text{_unit} )</td>
<td>→</td>
<td>( n\text{_unit} )</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'CEIL(symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{MTH} \quad \text{REAL} \quad \text{NXT} \quad \text{NXT} \quad \text{CEIL} \)

Affected by Flags: Numerical Results (-3)

Examples: \( 3.2 \) \text{CEIL} returns 4; \( -3.2 \) \text{CEIL} returns -3.

Related Commands: FLOOR, IP, RND, TRNC

CENTR

Center Command: Adjusts the first two parameters in the reserved variable \( PPAR, \langle x_{\text{min}}, y_{\text{min}} \rangle \) and \( \langle x_{\text{max}}, y_{\text{max}} \rangle \), so that the point represented by the argument \( \langle x, y \rangle \) is the plot center.
**CF**

Clear Flag Command: Clears the specified user or system flag.

### Keyboard Access:

- **MODES**  
- **FLAG**  
- **CF**
- **PRG**  
- **TEST**  
- **NXT**  
- **NXT**  
- **CF**

### Affected by Flags: None

### Remarks: User flags are numbered 1 through 64. System flags are numbered −1 through −64. See appendix C, “System Flags,” for a listing of HP 48 system flags and their flag numbers.

### Related Commands: FC?, FC?C, FS?, FS?C, SF
**CHOOSE**

**Create User-Defined Choose Box Command:** Creates a user-defined choose box.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;prompt&quot;</td>
<td>{ c_1 \ldots c_n }</td>
<td>\text{n}_{\text{pos}}</td>
<td>\text{obj or result}</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>&quot;prompt&quot;</td>
<td>{ c_1 \ldots c_n }</td>
<td>\text{n}_{\text{pos}}</td>
<td></td>
<td>&quot;0&quot;</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \text{PRG \; \text{NXT \; \text{IN \; CHOOS}}}

**Affected by Flags:** None

**Remarks:** CHOOSE creates a standard single-choice choose box based on the following specifications.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;prompt&quot;</td>
<td>A message that appears at the top of choose box. If &quot;prompt&quot; is an empty string (&quot;&quot;), no message is displayed.</td>
</tr>
<tr>
<td>{ c_1 \ldots c_n }</td>
<td>Definitions that appear within the choose box. A choice definition (c_x) can have two formats.</td>
</tr>
<tr>
<td>\text{n}_{\text{pos}}</td>
<td>The position number of an item definition. This item is highlighted when the choose box appears. If \text{n}_{\text{pos}}=0, no item is highlighted, and the choose box can be used to view items only.</td>
</tr>
</tbody>
</table>

If you choose an item from the choose box and press \text{OK}, CHOOSE returns the \text{result} (or the object itself if no result is
specified) to level 2 and 1 to level 1. If you press \textit{CANCEL}, \textit{CHOOSE} returns \emptyset. Also, if $n_{pos}=0$, \textit{CHOOSE} returns \emptyset.

**Example:** \textit{CHOOSE} with the following three lines on the display.

"Select a Program:"
{ "Pie Chart" «PIE» } { "Inverse Function" «ROOTR» } { "Animate Taylor" «TSA» } 

\texttt{1}

would produce the following display.

![Select a Program dialog box]

**Related Commands:** \texttt{INFORM}, \texttt{NOVAL}

---

**\%CH**

**Percent Change Function:** Returns the percent change from $x$ (level 2) to $y$ (level 1) as a percentage of $x$.

\[
\%CH(x,y) = \frac{100(y-x)}{x}
\]

\[
\%CH(x,'symb') = \%CH(symb,x)
\]

\[
\%CH(symb_1,symb_2) = \%CH(symb_1,x) = \%CH(x,symb_2)
\]

\[
\%CH(x_{\text{unit}},y_{\text{unit}}) = \frac{100(y_{\text{unit}}-x_{\text{unit}})}{x_{\text{unit}}}
\]

\[
\%CH(x_{\text{unit}},'symb') = \%CH('symb',x_{\text{unit}})
\]

\[
\%CH('symb',x_{\text{unit}}) = \%CH(x_{\text{unit}},'symb')
\]
%CH

Keyboard Access:  \( MTH \) \( \text{REAL} \) \( %CH \)

Affected by Flags:  Numerical Results (−3)

Remarks:  If both arguments are unit objects, the units must be consistent with each other. The dimensions of a unit object are dropped from the result, but units are part of the calculation.

For more information on using temperature units with arithmetic functions, refer to the keyword entry for +.

Examples:  1 m 500 cm \( %CH \) returns 400, because 500 cm represents an increase of 400% over 1 m.

\( 100_K \) \( 150_K \) \( %CH \) returns 50.

Related Commands:  %, %T

CHR

Character Command: Returns a string representing the HP 48 character corresponding to the character code \( n \).

\[
\begin{array}{ccc}
\text{Level 1} & \rightarrow & \text{Level 1} \\
n & \rightarrow & " \text{string}"
\end{array}
\]

Keyboard Access:
\( \text{CHARS} \) \( \text{CHR} \)

PRG TYPE NXT CHR

Affected by Flags:  None

Remarks:  The character codes are an extension of ISO 8859/1. Codes 128 through 159 are unique to the HP 48. See the entry for NUM for a complete list of characters and character codes.

The default character \( \# \) is supplied for all character codes that are not part of the normal HP 48 display character set.
Character code 0 is used for the special purpose of marking the end of the command line. Attempting to edit a string containing this character causes the error Can't Edit CHR(0).

You can use the CHARS application to find the character code for any character used by the HP 48. See "Keying in Special Characters" in chapter 2 of the HP 48 User’s Guide.

Related Commands: NUM, POS, REPL, SIZE, SUB

CKSM

Checksum Command: Specifies the error-detection scheme.

\[
\begin{array}{c|c}
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
n_{\text{checksum}} & \rightarrow & \\
\end{array}
\]

Keyboard Access: (I/O) IOPAR CKSM

Affected by Flags: None

Remarks: Legal values for \( n_{\text{checksum}} \) are as follows:

- 1: 1-digit arithmetic checksum.
- 2: 2-digit arithmetic checksum.
- 3: 3-digit cyclic redundancy check (default).

The CKSM specified is the error-detection scheme that will be requested by KGET, PKT, or SEND. If the sender and receiver disagree about the request, however, then a 1-digit arithmetic checksum will be used.

IR transmission should use checksum type 3.

Related Commands: BAUD, PARITY, TRANSIO
CLEAR

Clear Command: Removes all objects from the stack.

<table>
<thead>
<tr>
<th>Level n . . . Level 1</th>
<th>→</th>
<th>Level n . . . Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( obj_n \ldots obj_1 )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( \leftrightarrow \) CLEAR

Affected by Flags: None

Remarks: To recover a cleared stack, press \( \leftarrow \) (UNDO) before executing any other operation. There is no programmable command to recover the stack.

Related Commands: CLVAR, PURGE

CLKADJ

Adjust System Clock Command: Adjusts the system time by \( z \) clock ticks, where 8192 clock ticks equal 1 second.

\[
\{ \}

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( \leftrightarrow \) TIME NXT NXT CLKADJ

Affected by Flags: None

Remarks: If \( x \) is positive, \( x \) clock ticks are added to the system time. If \( x \) is negative, \( x \) clock ticks are subtracted from the system time.

Example: \(-20480 \) CLKADJ decrements the system time by 2.5 seconds.
CLOSEIO

Related Commands: —TIME

CLLCD

Clear LCD Command: Clears (blanks) the stack display.

Keyboard Access: PRG NXT OUT CLLCD

Affected by Flags: None

Remarks: The menu labels continue to be displayed after execution of CLLCD.

When executed from a program, the blank display persists only until the keyboard is ready for input. To cause the blank display to persist until a key is pressed, execute FREEZE after executing CLLCD.
(When executed from the keyboard, CLLCD automatically freezes the display.)

Example: Evaluating « CLLCD 7 FREEZE » blanks the display (except the menu labels), then freezes the entire display.

Related Commands: DISP, FREEZE

CLOSEIO

Close I/O Port Command: Closes the serial port and the IR port, and clears the input buffer and any error messages for KERRM.

Keyboard Access: (1/0) CLOSE

Affected by Flags: None

Remarks: When the HP 48 turns off, it automatically closes the serial and IR ports, but does not clear KERRM. Therefore, CLOSEIO is not needed to close the ports, but can conserve power without turning off the calculator.

Executing HP 48 Kermit protocol commands automatically clears the input buffer; however, executing non-Kermit commands (such as SRECV and XMIT) does not.
CLOSEIO

CLOSEIO also clears error messages from KERRM. This can be useful when debugging.

Related Commands:  BUFLEN, OPENIO

CL*$

Clear Sigma Command:  Purges the current statistics matrix (reserved variable $DAT$).

Keyboard Access:  $\leftarrow$STAT DATA $\leftarrow$CL*$

Affected by Flags:  None

Related Commands:  RCL*$, STO*$, $+, -$

CLTEACH

Clear Teaching Examples Command:  Removes the EXAMPLES subdirectory and its contents from the HOME directory.

Keyboard Access:  None. Must be typed in.

Affected by Flags:  None

Related Commands:  TEACH

CLUSR

Clear Variables Command:  Provided for compatibility with the HP 28. CLUSR is the same as CLVAR. See CLVAR.
**CLVAR**

**Clear Variables Command:** Purges all variables and empty subdirectories in the current directory.

**Keyboard Access:** None. Must be typed in.

**Affected by Flags:** None

**Related Commands:** CLUSR, PGDIR, PURGE

---

**CNRM**

**Column Norm Command:** Returns the column norm (one-norm) of the array argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ array ]</td>
<td>( x_{\text{column norm}} )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH MATR NORM CNRM]

**Affected by Flags:** None

**Remarks:** The column norm of a matrix is the maximum (over all columns) of the sum of the absolute values of all elements in each column. For a vector, the column norm is the sum of the absolute values of the vector elements. For complex arrays, the absolute value of a given element \((x, y)\) is \(\sqrt{x^2 + y^2}\).

**Related Commands:** CROSS, DET, DOT, RNRM
→COL

Matrix to Columns Command: Transforms a matrix into a series of column vectors and returns the vectors and a column count, or transforms a vector into its elements and returns the elements and an element count.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level n+1 ...</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ matrix ]</td>
<td></td>
<td>[ vector ]_col1</td>
<td>[ vector ]_coln</td>
</tr>
<tr>
<td>[ vector ]</td>
<td></td>
<td>element_1</td>
<td>element_n</td>
</tr>
</tbody>
</table>

Keyboard Access: \( 	ext{MTH} \ M \	ext{ATR} \ COL \rightarrow COL \)

Affected by Flags: None

Remarks: →COL introduces no rounding error.

Related Commands: COL→, →ROW, ROW→

COL+

Insert Column Command: Inserts an array (vector or matrix) into a matrix (or one or more elements into a vector) at the position indicated by \( n_{\text{index}} \), and returns the modified array.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ matrix ]_1</td>
<td>[ matrix ]_2</td>
<td>( n_{\text{index}} ) → [ matrix ]_3</td>
<td></td>
</tr>
<tr>
<td>[ matrix ]_1</td>
<td>[ vector ]_column</td>
<td>( n_{\text{index}} ) → [ matrix ]_2</td>
<td></td>
</tr>
<tr>
<td>[ vector ]_1</td>
<td>( n_{\text{element}} )</td>
<td>( n_{\text{index}} ) → [ vector ]_2</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( 	ext{MTH} \ M \	ext{ATR} \ COL \ COL+ \)

Affected by Flags: None

3-54 Command Reference
Remarks: The inserted array must have the same number of rows as the target array.

\( n_{\text{index}} \) is rounded to the nearest integer. The original array is redimensioned to include the new columns or elements, and the elements at and to the right of the insertion point are shifted to the right.

Related Commands: COL-, CSWP, ROW+, ROW-

---

**COL-**

Delete Column Command: Deletes column \( n \) of a matrix (or element \( n \) of a vector), and returns the modified matrix (or vector) and the deleted column (or element).

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([ [\text{matrix}]<em>1 ), \text{n}</em>{\text{column}})</td>
<td>([ [\text{matrix}]<em>2 ), \text{n}</em>{\text{column}})</td>
<td>→</td>
<td>([ [\text{vector}]<em>1 ), \text{n}</em>{\text{element}})</td>
<td>([ [\text{vector}]_2 ), \text{element}_n)</td>
</tr>
</tbody>
</table>

Keyboard Access: `MTH MATR COL COL-`

Affected by Flags: None

Remarks: \( n \) is rounded to the nearest integer.

Related Commands: COL+, CSWP, ROW+, ROW-
**COL**

**Columns to Matrix Command:** Transforms a series of column vectors and a column count into a matrix containing those columns, or transforms a sequence of numbers and an element count into a vector with those numbers as elements.

<table>
<thead>
<tr>
<th>Level n+1</th>
<th>...</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[vector]_{column_1}</td>
<td>[vector]_{column_n}</td>
<td>n_{column_count}</td>
<td>→</td>
<td>[[matrix]]</td>
<td></td>
</tr>
<tr>
<td>element_1</td>
<td>element_n</td>
<td>n_{element_count}</td>
<td>→</td>
<td>[vector]</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] MATR COL COL

**Affected by Flags:** None

**Remarks:** All vectors must have the same length. The column or element count is rounded to the nearest integer.

**Related Commands:** →COL, →ROW, ROW

---

**COLCT**

**Collect Like Terms Command:** Simplifies an algebraic expression or equation by “collecting” like terms.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb_1'</td>
<td>→</td>
<td>'symb_2'</td>
</tr>
<tr>
<td>x</td>
<td>→</td>
<td>x</td>
</tr>
<tr>
<td>(x, y)</td>
<td>→</td>
<td>(x, y)</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] SYMBOLIC COLCT

**Affected by Flags:** None

---

3-56 Command Reference
Remarks: COLCT operates separately on the two sides of an equation, so that like terms on opposite sides of the equation are not combined.

Examples: '6+EXP(10)' COLCT returns 8.71828182846.
'5*X+9' COLCT returns '14*X'.
'X+1_m+X+9_cm' COLCT returns '(109_cm)*X'.
'X^Z*Y*X^T*Y' COLCT returns 'X^(T+Z)*Y^2'.
'X+3*X+Y+Y' COLCT returns '4*X+2*Y'.

Related Commands: EXPAN, ISOL, QUAD, SHOW

\[ \text{COLΣ} \]

Sigma Columns Command: Specifies the independent-variable and dependent-variable columns of the current statistics matrix (the reserved variable \(Σ\ DAT\)).

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_{xcol})</td>
<td>(x_{ycol})</td>
<td>→</td>
<td>()</td>
</tr>
</tbody>
</table>

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: COLΣ combines the functionality of XCOL and YCOL. It is included in the HP 48 for compatibility with the HP 28S.

The independent-variable column number \(x_{xcol}\) is stored as the first parameter in the reserved variable \(Σ\ PAR\) (the default is 1). The dependent-variable column number \(x_{ycol}\) is stored as the second parameter in \(Σ\ PAR\) (the default is 2).

COLΣ accepts and stores noninteger real numbers, but subsequent commands that use these two parameters in \(Σ\ PAR\) will cause errors.
Example: \( \text{COL}_2 \) sets column 2 in \( \Sigma DAT \) as the independent-variable column, sets column 5 as the dependent-variable column, and stores 2 and 5 as the first and second elements in \( \Sigma PAR \).

Related Commands: BARPLOT, BESTFIT, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL, YCOL

**COMB**

Combinations Function: Returns the number of possible combinations of \( n \) items taken \( m \) at a time.

\[
\binom{n}{m} = \frac{n!}{m!(n-m)!}
\]

The arguments \( n \) and \( m \) must each be less than \( 10^{12} \).

Related Commands: PERM, !
**CON**

**Constant Array Command:** Returns a constant array, defined as an array whose elements all have the same value.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ n_{columns} }</td>
<td>z_{constant}</td>
<td>→</td>
<td>[ \text{vector}_{constant} ]</td>
</tr>
<tr>
<td>{ n_{rows} m_{columns} }</td>
<td>z_{constant}</td>
<td>→</td>
<td>[ \text{matrix}_{constant} ]</td>
</tr>
<tr>
<td>[ R-array ]</td>
<td>x_{constant}</td>
<td>→</td>
<td>[ \text{R-array}_{constant} ]</td>
</tr>
<tr>
<td>[ C-array ]</td>
<td>z_{constant}</td>
<td>→</td>
<td>[ \text{C-array}_{constant} ]</td>
</tr>
<tr>
<td>'name'</td>
<td>z_{constant}</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [MATR] [MAKE] [CON]

**Affected by Flags:** None

**Remarks:** The constant value is a real or complex number taken from level 1. The resulting array is either a new array, or an existing array with its elements replaced by the constant, depending on the object in level 2.

- **Creating a new array:** If level 2 contains a list of one or two integers, \( \text{CON} \) returns a new array. If the list contains a single integer \( n_{columns} \), \( \text{CON} \) returns a constant vector with \( n \) elements. If the list contains two integers \( n_{rows} \) and \( m_{columns} \), \( \text{CON} \) returns a constant matrix with \( n \) rows and \( m \) columns.

- **Replacing the elements of an existing array:** If level 2 contains an array, \( \text{CON} \) returns an array of the same dimensions, with each element equal to the constant. If the constant is a complex number, the original array must also be complex.

If level 2 contains a name, the name must identify a variable that contains an array. In this case, the elements of the array are replaced by the constant. If the constant is a complex number, the original array must also be complex.

**Examples:**
- \{2 \ 2 \} \ 6 \ \text{CON} \ returns \ the \ matrix \ [ [ 6 \ 6 ] [ 6 \ 6 ] ].
- \( \{2, 4\} \ (7, 9) \) \ 3 \ \text{CON} \ returns \ the \ complex \ vector \ [ (3, 0) \ (3, 0) ] .
CON

Related Commands: IDN

COND

Condition Number Command: Returns the 1-norm (column norm) condition number of a square matrix.

Level 1 \( \begin{array}{c} \rightarrow \end{array} \) Level 1

\[
[\text{matrix}]_{n \times n} \rightarrow x_{\text{condition number}}
\]

Keyboard Access: \[ \text{MTH} \text{ MTRN} \text{ NORM} \text{ COND} \]

Affected by Flags: None

Remarks: The condition number of a matrix is the product of the norm of the matrix and the norm of the inverse of the matrix. COND uses the 1-norm and computes the condition number of the matrix without computing the inverse of the matrix.

The condition number expresses the sensitivity of the problem of solving a system of linear equations having coefficients represented by the elements of the matrix (this includes inverting the matrix). That is, it indicates how much an error in the inputs may be magnified in the outputs of calculations using the matrix.

In many linear algebra computations, the base 10 logarithm of the condition number of the matrix is an estimate of the number of digits of precision that might be lost in computations using that matrix. A reasonable rule of thumb is that the number of digits of accuracy in the result is approximately \( \text{MIN}(12, 15 - \log_{10}(|\text{COND}|)) \).

Example: The following program computes the above rule of thumb for the number of accurate digits:

\[
\begin{array}{c}
\text{DUP SIZE 1 GET LOG SWAP COND LOG + 11 SWAP -}
\end{array}
\]
Related Commands: SNRM, SRAD, TRACE

## CONIC

**Conic Plot Type Command:** Sets the plot type to CONIC.

**Keyboard Access:** («)(PLOT)F1

**Affected by Flags:** None

**Remarks:** When the plot type is CONIC, the DRAW command plots the current equation as a second-order polynomial of two real variables. The current equation is specified in the reserved variable EQ. The plotting parameters are specified in the reserved variable PPAR, which has this form:

\[
\langle x_{\text{min}}, y_{\text{min}} \rangle \langle x_{\text{max}}, y_{\text{max}} \rangle \text{ indep res axes ptype depend } \n\]

For plot type CONIC, the elements of PPAR are used as follows:

- \( \langle x_{\text{min}}, y_{\text{min}} \rangle \) is a complex number specifying the lower left corner of PICT (the lower left corner of the display range). The default value is \((-6.5, -3.1)\).

- \( \langle x_{\text{max}}, y_{\text{max}} \rangle \) is a complex number specifying the upper right corner of PICT (the upper right corner of the display range). The default value is \((6.5, 3.2)\).

- \text{indep} is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value is \(X\).

- \text{res} is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.

- \text{axes} is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes, or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is \((0,0)\).
CONIC

- `ptype` is a command name specifying the plot type. Executing the command CONIC places the command name CONIC in PPAR.

- `depend` is a name specifying the dependent variable. The default value is Y.

The current equation is used to define a pair of functions of the independent variable. These functions are derived from the second-order Taylor's approximation to the current equation. The minimum and maximum values of the independent variable (the plotting range) can be specified in `indep`; otherwise, the values in \( (x_{\text{min}}, y_{\text{min}}) \) and \( (x_{\text{max}}, y_{\text{max}}) \) (the display range) are used. Lines are drawn between plotted points unless flag –31 is set.

**Related Commands:** BAR, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

---

CONJ

**Conjugate Analytic Function:** Conjugates a complex number or a complex array.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>→</td>
<td>( x )</td>
</tr>
<tr>
<td>( (x, y) )</td>
<td>→</td>
<td>( (x, -y) )</td>
</tr>
<tr>
<td>([ R-array ])</td>
<td>→</td>
<td>([ R-array ])</td>
</tr>
<tr>
<td>([ C-array ]_1)</td>
<td>→</td>
<td>([ C-array ]_2)</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'CONJ(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH NXT CMPL NXT CONJ

**Affected by Flags:** Numerical Results (–3)
Remarks: Conjugation is the negation (sign reversal) of the imaginary part of a complex number. For real numbers and real arrays, the conjugate is identical to the original argument.

Example: \[
\begin{bmatrix}
3 & 4 \\
7 & 2
\end{bmatrix}
\]
CONJ returns \[
\begin{bmatrix}
3 & -4 \\
7 & -2
\end{bmatrix}
\].

A square matrix \(A\) containing complex elements is said to be Hermitian if \(A^H = A\), where \(A^H\) is the same as a normal transpose except that the complex conjugate of each element is used. The following program returns 1 if the input matrix is Hermitian, and a 0 if it is not.

\[
\text{« DUP TRN CONJ SAME »}
\]

Related Commands: ABS, IM, RE, SCONJ, SIGN

---

**CONLIB**

Open Constants Library Command: Opens the Constants Library catalog.

Keyboard Access: \(\leftarrow\text{EQ LIB}\colon\text{COLIB CONLI}\)

Affected by Flags: None

Related Commands: CONST

---

**CONST**

Constant Value Command: Returns the value of a constant.

\[
\begin{array}{c|c}
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
\text{\texttt{name}} & \rightarrow & x
\end{array}
\]

Keyboard Access: \(\leftarrow\text{EQ LIB}\colon\text{COLIB CONST}\)

Affected by Flags: Units Type (60), Units Usage (61)
**CONST**

**Remarks:** CONST returns the value of the specified constant. It chooses the unit type depending on flag 60 (SI if clear, English if set), and uses the units depending on flag 61 (uses units if clear, no units if set).

See “Using the Constants Library” in chapter 25 of the *HP 48 User’s Guide* for a list of the constants available in the HP 48’s Constants Library.

**Related Commands:** CONLIB

---

**CONT**

**Continue Program Execution Command:** Resumes execution of a halted program.

**Keyboard Access:** («)(CONT)

**Affected by Flags:** None

**Remarks:** Since CONT is a command, it can be assigned to a key or to a custom menu.

**Example:** The program

```
"Enter A, press ( CONT )" ( CONT ) MENU PROMPT
```

displays a prompt message, builds a menu with the CONT command assigned to the first menu key, and halts the program for data input. After entering data, pressing CONT resumes program execution. (Note that pressing («)(CONT) is equivalent to pressing CONT.)

**Related Commands:** HALT, KILL, PROMPT
CONVERT

Convert Units Command: Converts a source unit object to the dimensions of a target unit.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)_{units\text{source}}</td>
<td>(x_2)_{units\text{target}}</td>
<td>→</td>
<td>(x_3)_{units\text{target}}</td>
</tr>
</tbody>
</table>

Keyboard Access:  

Affected by Flags: None

Remarks: The source and target units must be compatible. The number part \(x_2\) of the target unit object is ignored.

Related Commands: UBASE, UFACT, →UNIT, UVAL

CORR

Correlation Command: Returns the correlation coefficient of the independent and dependent data columns in the current statistics matrix (reserved variable \(\Sigma\)DAT).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>(x_{\text{correlation}})</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access:  

Affected by Flags: None

Remarks: The columns are specified by the first two elements in the reserved variable \(\Sigma\)PAR, set by XCOL and YCOL, respectively. If \(\Sigma\)PAR does not exist, CORR creates it and sets the elements to their default values (1 and 2).
CORR

The correlation is computed with the following formula:

$$\frac{\sum_{i=1}^{n}(x_{in1} - \bar{x}_{n1})(x_{in2} - \bar{x}_{n2})}{\sqrt{\sum_{i=1}^{n}(x_{in1} - \bar{x}_{n1})^2 \sum_{i=1}^{n}(x_{in2} - \bar{x}_{n2})^2}}$$

where $x_{in1}$ is the $i$th coordinate value in column $n_1$, $x_{in2}$ is the $i$th coordinate value in the column $n_2$, $\bar{x}_{n1}$ is the mean of the data in column $n_1$, $\bar{x}_{n2}$ is the mean of the data in column $n_2$, and $n$ is the number of data points.

**Related Commands:** COLΣ, COV, PREDX, PREDY, XCOL, YCOL

---

COS

**Cosine Analytic Function:** Returns the cosine of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>$\rightarrow$</td>
<td>$\cos z$</td>
</tr>
<tr>
<td>'symb'</td>
<td>$\rightarrow$</td>
<td>'COS(symb)'</td>
</tr>
<tr>
<td>$x_{\text{unit}_\text{angular}}$</td>
<td>$\rightarrow$</td>
<td>$\cos (x_{\text{unit}_\text{angular}})$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [COS]

**Affected by Flags:** Numerical Results (−3), Angle Mode (−17, −18)

**Remarks:** For real arguments, the current angle mode determines the number’s interpretation as an angle, unless the angular units are specified.

For complex arguments, $\cos(x + iy) = \cos x \cosh y - i \sin x \sinh y$.

If the argument for COS is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing COS with a unit object, the angle mode must be set to Radians (since this is a “neutral” mode).
COSH

Hyperbolic Cosine Analytic Function: Returns the hyperbolic cosine of the argument.

Keyboard Access: MTH HYP COSH

Affected by Flags: Numerical Results (-3)

Remarks: For complex arguments, \( \cosh(x + iy) = \cosh x \cos y + i \sinh x \sin y \).

Related Commands: ACOSH, SINH, TANH

COV

Covariance Command: Returns the sample covariance of the independent and dependent data columns in the current statistics matrix (reserved variable \( \Sigma DAT \)).

Keyboard Access: STAT FIT COV
**COV**

**Affected by Flags:** None

**Remarks:** The columns are specified by the first two elements in reserved variable ΣPAR, set by XCOL and YCOL respectively. If ΣPAR does not exist, COV creates it and sets the elements to their default values (1 and 2).

The covariance is calculated with the following formula:

$$\frac{1}{n-1} \sum_{i=1}^{n} (x_{in_1} - \bar{x}_{n_1})(x_{in_2} - \bar{x}_{n_2})$$

where $x_{in_1}$ is the $i$th coordinate value in column $n_1$, $x_{in_2}$ is the $i$th coordinate value in the column $n_2$, $\bar{x}_{n_1}$ is the mean of the data in column $n_1$, $\bar{x}_{n_2}$ is the mean of the data in column $n_2$, and $n$ is the number of data points.

**Related Commands:** COLΣ, CORR, PCOV, PREDX, PREDY, XCOL, YCOL

---

**CR**

**Carriage Right Command:** Prints the contents, if any, of the printer buffer.

**Keyboard Access:**  

**Affected by Flags:** Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

**Remarks:** When using the HP 82240B Infrared Printer (flag −34 clear), CR leaves the printhead on the right end of the just printed line.

When printing to the serial port (flag −34 set), CR sends to the printer a string that encodes the line termination method. The default termination method is carriage-return/linefeed. The string is the fourth parameter in the reserved variable PRTPAR.

**Related Commands:** DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1
CRDIR

Create Directory Command: Creates an empty subdirectory with the specified name within the current directory.

Level 1  →  Level 1
'global'  →

Keyboard Access:  ←(MEMORY) DIR CRDIR

Affected by Flags:  None

Remarks:  CRDIR does not change the current directory; evaluate the name of the new subdirectory to make it the current directory.

Related Commands:  HOME, PATH, PGDIR, UPDIR

CROSS

Cross Product Command: CROSS returns the cross product $C = A \times B$ of vectors $A$ and $B$.

Level 2  Level 1  →  Level 1

Keyboard Access:  MTH VECTR CROSS

Affected by Flags:  None

Remarks:  The arguments must be vectors having two or three elements, and do not both need the same number of elements. (The HP 48 automatically converts a two-element argument $[d_1 \ d_2]$ to a three-element argument $[d_1 \ d_2 \ \emptyset]$.)
CROSS

Related Commands: CNRM, DET, DOT, RNRM

CSWP

Column Swap Command: Swaps columns $i$ and $j$ of the argument matrix and returns the modified matrix, or swaps elements $i$ and $j$ of the argument vector and returns the modified vector.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[[ matrix ]]_1$</td>
<td>$n_{column i}$</td>
<td>$n_{column j}$</td>
<td>→</td>
<td>$[[ matrix ]]_2$</td>
</tr>
<tr>
<td>$[ vector ]_1$</td>
<td>$n_{element i}$</td>
<td>$n_{element j}$</td>
<td>→</td>
<td>$[ vector ]_2$</td>
</tr>
</tbody>
</table>

Keyboard Access: MTH MATRX COL CSWP

Affected by Flags: None

Remarks: Column numbers are rounded to the nearest integer. Vector arguments are treated as row vectors.

Related Commands: COL+, COL−, RSWP

CYLIN

Cylindrical Mode Command: Sets Cylindrical coordinate mode.

Keyboard Access:

MTH VECTR NXT CYLIN

Affected by Flags: None

Remarks: CYLIN clears flag −15 and sets flag −16, and displays the R∠Z annunciator.
In Cylindrical mode, vectors are displayed as polar components. Therefore, a 3D vector would appear as $[ R \Delta \theta \ Z ]$.

**Related Commands:** RECT, SPHERE

---

### C→PX

**Complex to Pixel Command:** Converts the specified user-unit coordinates to pixel coordinates.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
</table>
| $(x, y)$      | → | $\{ \#n \#m \}$

**Keyboard Access:** PICT  NXT  C→PX

**Affected by Flags:** None

**Remarks:** The user-unit coordinates are derived from the $(x_{\text{min}}, y_{\text{min}})$ and $(x_{\text{max}}, y_{\text{max}})$ parameters in the reserved variable PPAR.

**Related Commands:** PX→C

---

### C→R

**Complex to Real Command:** Separates the real and imaginary parts of a complex number or complex array.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x, y)$</td>
<td>→</td>
<td>$x$</td>
<td>$y$</td>
</tr>
<tr>
<td>[ C-array ]</td>
<td>→</td>
<td>[ R-array ]_1</td>
<td>[ R-array ]_2</td>
</tr>
</tbody>
</table>

---

*Command Reference 3-71*
C→R

Keyboard Access:

```
MTH NXT CMPL C→R
PRG TYPE NXT C→R
```

Affected by Flags: None

Remarks: The result in level 2 represents the real part of the complex argument. The result in level 1 represents the imaginary part of the complex argument.

Related Commands: R→C, RE, IM

---

DARCY

Darcy Friction Factor Function: Calculates the Darcy friction factor of certain fluid flows.

```
\begin{array}{ccc}
\text{Level 2} & \text{Level 1} & \rightarrow & \text{Level 1} \\
x_{e/D} & y_{Re} & \rightarrow & x_{Darcy}
\end{array}
```

Keyboard Access: \(\equiv\) Eq Lib UTILS DARCY

Affected by Flags: None

Remarks: DARCY calculates the Fanning friction factor and multiplies it by 4. \(x_{e/D}\) is the relative roughness—the ratio of the conduit roughness to its diameter. \(y_{Re}\) is the Reynolds number. The function uses different computation routines for laminar flow (\(Re \leq 2100\)) and turbulent flow (\(Re > 2100\)). \(x_{e/D}\) and \(y_{Re}\) must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Related Commands: FANNING
# DATE

## Date Command:
Returns the system date to level 1.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>date</td>
</tr>
</tbody>
</table>

**Keyboard Access:** ⌚️ DATE

**Affected by Flags:** Date Format (—42)

**Example:** If the current date is May 12, 1990, if flag —42 is clear, and if the display mode is Standard, DATE returns 5.12.1990. (The trailing zeros are dropped.)

**Related Commands:** DATE+, D DAYS, TIME, TSTR

## →DATE

## Set Date Command:
Sets the system date to date.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** ⌚️ →DATE

**Affected by Flags:** Date Format (—42)

**Remarks:** date has the form MM.DDYYYY or DD.MMYYYY, depending on the state of flag —42. MM is month, DD is day, and YYYY is year. If YYYY is not supplied, the current specification for the year is used. The range of allowable dates is January 1, 1991 to December 31, 2090.
→DATE

Example: If flag -42 is set and the current system year is 1995, then 
28.07 DATE sets the system date as July 28, 1995.

Related Commands: →TIME

DATE+

Date Addition Command: Returns a past or future date, given a 
date in level 2 and a number of days in level 1.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>date1</td>
<td>x_days</td>
<td>→</td>
<td>date_new</td>
</tr>
</tbody>
</table>

Keyboard Access: TIME NXT DATE+

Affected by Flags: Date Format (-42)

Remarks: If x_days is negative, DATE+ calculates a past date. The 
range of allowable dates is October 15, 1582, to December 31, 9999.

Related Commands: DATE, D DAYS

DBUG

Debug Operation: Starts program execution, then suspends it as if 
HALT were the first program command.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>« program » or 'program name'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: PRG NXT RUN DEBUG
**DEC**

**Affected by Flags:** None

**Remarks:** DBUG is not programmable.

**Related Commands:** HALT, NEXT, SST, SST

---

**DDAYS**

**Delta Days Command:** Returns the number of days between two dates.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{date}_1$</td>
<td>$\text{date}_2$</td>
<td>→</td>
<td>$x_{\text{days}}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\leftarrow\text{TIME} \rightarrow \text{NEXT} \ DDAYS\)

**Affected by Flags:** Date Format \((-42)\)

**Remarks:** If the level 2 date is chronologically later than the level 1 date, the result is negative. The range of allowable dates is October 15, 1582, to December 31, 9999.

**Related Commands:** DATE, DATE+

---

**DEC**

**Decimal Mode Command:** Selects decimal base for binary integer operations. (The default base is decimal.)

**Keyboard Access:** \(\text{MTH} \ \text{BASE} \ \text{DEC}\)

**Affected by Flags:** Binary Integer Wordsize \((-5\ \text{through} \ -10)\), Binary Integer Base \((-11, -12)\)

**Remarks:** Binary integers require the prefix \#. Binary integers entered and returned in decimal base automatically show the suffix \(\dagger\). If the current base is not decimal, then you can enter a decimal
DEC

number by ending it with \texttt{d}. It will be displayed in the current base when it is entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Related Commands: BIN, HEX, OCT, RCWS, STWS

DECR

Decrement Command: Takes a variable on level 1, subtracts 1, stores the new value back into the original variable, and returns the new value to level 1.

\begin{tabular}{|c|c|}
\hline
Level 1 & \rightarrow & Level 1 \\
\hline
\texttt{name} & \rightarrow & x_{\text{new}} \\
\hline
\end{tabular}

Keyboard Access: \texttt{MEMORY ARITH DECR}

Affected by Flags: None

Remarks: The contents of \textit{name} must be a real number.

Example: If 35.7 is stored in \texttt{A}, \texttt{A DECR} returns 34.7.

The following program counts down from 100 to 0 and leaves the integers 100 to 0 on the stack:

\begin{verbatim}
«
100 \texttt{A} STO
\texttt{WHILE A REPEAT \texttt{A} DECR END}
\texttt{A PURGE}
»
\end{verbatim}

Related Commands: INCR
Define Variable or Function Command: Stores the expression on the right side of the = in the variable specified on the left side, or creates a user-defined function.

\[
\begin{array}{|c|c|}
\hline
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
'name=exp' & \rightarrow & \\
'name(name_1 \ldots name_n)=exp(name_1 \ldots name_n)' & \rightarrow & \\
\hline
\end{array}
\]

Keyboard Access: \( \leftarrow \) (DEF)

Affected by Flags: Numerical Results \((-3)\)

For arguments of the form 'name=exp', if flag -3 is set, expression will be evaluated to a number before it is stored in name. (If exp contains a formal variable, DEFINE will error if flag -3 is set.)

Remarks: If the left side of the equation is name only, DEFINE stores exp in the variable name.

If the left side of the equation is name followed by parenthetical arguments name_1 \ldots name_n, DEFINE creates a user-defined function and stores it in the variable name.

Examples: 'A=2*X' DEFINE stores '2*X' in variable A.

'A(X,Y)=2*X+3/Y' DEFINE creates a user-defined function A. The contents of A is the program « → X Y '2*X+3/Y' ».

Related Commands: STO
**DEG**

**Degrees Command:** Sets Degrees angle mode.

**Keyboard Access:** \( \rightarrow \text{MODES} \uparrow \text{ANG} \downarrow \text{DEG} \)

**Affected by Flags:** None

**Remarks:** DEG clears flags –17 and –18, and clears the RAD and GRAD annunciators.

In Degrees angle mode, real-number arguments that represent angles are interpreted as degrees, and real-number results that represent angles are expressed in degrees.

**Related Commands:** GRAD, RAD

---

**DELALARM**

**Delete Alarm Command:** Deletes the alarm specified in level 1.

\[
\begin{array}{|c|c|}
\hline
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
n_{\text{index}} & \rightarrow & \\
\hline
\end{array}
\]

**Keyboard Access:** \( \rightarrow \text{TIME} \text{ALRM} \text{DELAL} \)

**Affected by Flags:** None

**Remarks:** If \( n_{\text{index}} \) is 0, all alarms in the system alarm list are deleted.

**Related Commands:** FINDALARM, RCLALARM, STOALARM
Delay Command: Specifies how many seconds the HP 48 waits between sending lines of information to the printer.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{\text{delay}}$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \(\leftarrow\)(I/O) PRINT PRTPAR DELAY

Affected by Flags: Printing Device \((-34)\) and I/O Device \((-33)\)

Setting flag \(-34\) directs printer output to the serial port. In this case, flag \(-33\) must be clear.

If flag \(-34\) is set and transmit pacing is enabled (nonzero) in reserved variable $IOPAR$, then XON/XOFF handshaking controls data transmission and the delay setting has no effect.

Remarks: $x_{\text{delay}}$ specifies the delay time in seconds. The default delay is 1.8 seconds. The maximum delay is 6.9 seconds. (The sign of $x_{\text{delay}}$ is ignored, so \(-4\) DELAY is equivalent to \(4\) DELAY.)

The delay setting is the first parameter in the reserved variable PRTPAR.

A shorter delay setting can be useful when the HP 48 sends multiple lines of information to your printer (for example, when printing a program). To optimize printing efficiency, set the delay just longer than the time the printhead requires to print one line of information.

If you set the delay shorter than the time to print one line, you may lose information. Also, as the batteries in the printer lose their charge, the printhead slows down, and, if you have previously decreased the delay, you may have to increase it to avoid losing information. (Battery discharge will not cause the printhead to slow to more than the 1.8-second default delay setting.)

Related Commands: CR, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PRI
Delete Key Assignments Command: Clears user-defined key assignments.

### Level 1

<table>
<thead>
<tr>
<th>x_key</th>
<th>→</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_{key1} \ldots x_{keyn}}</td>
<td>→</td>
</tr>
<tr>
<td>0</td>
<td>→</td>
</tr>
<tr>
<td>'S'</td>
<td>→</td>
</tr>
</tbody>
</table>

Keyboard Access: («)(MODES) kE

Affected by Flags: User-Mode Lock (—61) and User Mode (—62) affect the status of the user keyboard.

Remarks: The argument $x_{key}$ is a real number $rc.p$ specifying the key by its row number, its column number, and its plane (shift). For a definition of plane, see ASN.

Specifying $\emptyset$ for $x_{key}$ clears all user key assignments and restores the standard key assignments.

Specifying $S$ as the argument for DELKEYS suppresses all standard key assignments on the user keyboard. This makes keys without user key assignments inactive on the user keyboard. (You can make exceptions using ASN, or restore them all using STOKEYS.) If you are stuck in User mode—probably with a “locked” keyboard—because you have reassigned or suppressed the keys necessary to cancel User mode, do a system halt (“warm start”): press and hold [ON] and the C key simultaneously, releasing the C key first. This cancels User mode.

Deleted user key assignments still take up from 2.5 to 15 bytes of memory each. You can free this memory by packing your user key assignments by executing RCLKEYS $\emptyset$ DELKEYS STOKEYS.

Related Commands: ASN, RCLKEYS, STOKEYS
DEPND

Dependent Variable Command: Specifies the dependent variable (and its plotting range for TRUTH plots).

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'global'</td>
<td>→</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{ global }</td>
<td>→</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{ global y_start y_end }</td>
<td>→</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{ y_start y_end }</td>
<td>→</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_start</td>
<td>→</td>
<td>y_end</td>
<td>→</td>
</tr>
</tbody>
</table>

Keyboard Access: [PLOT] PPAR DEPN

Affected by Flags: None

Remarks: The specification for the dependent variable name and its plotting range is stored in the reserved variable PPAR as follows:

- If the argument is a global variable name, that name replaces the dependent variable entry in PPAR.
- If the argument is a list containing a global name, that name replaces the dependent variable name but leaves unchanged any existing plotting range.
- If the argument is a list containing a global name and two real numbers, or a list containing a name, array, and real number, that list replaces the dependent variable entry.
- If the argument is a list containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the dependent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is Y.

The plotting range for the dependent variable is meaningful only for plot type TRUTH, where it restricts the region for which the equation
DEPND

is tested, and for plot type DIFFEQ, where it specifies the initial solution value and absolute error tolerance.

Related Commands: INDEP

DEPTH

Depth Command: Returns a real number representing the number of objects present on the stack (before DEPTH was executed).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>n</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \rightarrow \) \( \text{STACK} \) DEPTH

Affected by Flags: None

DET

Determinant Function: Returns the determinant of a square matrix.

\[ \{ \text{matrix} \} \rightarrow x_{\text{determinant}} \]

Keyboard Access: \( \text{MTH} \) MATR NORM \( \text{NXT} \) DET

Affected by Flags: Tiny Element (-54)

Remarks: The argument matrix must be square. DET computes the determinant of 1x1 and 2x2 matrices directly from the defining expression for the determinant. DET computes the determinant of a larger matrix by computing the Crout LU decomposition of the
matrix and accumulating the product of the decomposition's diagonal elements.

Since floating-point division is used to do this, the computed determinant of an integer matrix is often not an integer, even though the actual determinant of an integer matrix must be an integer. DET corrects this by rounding the computed determinant to an integer value. This technique is also used for noninteger matrices with determinants having fewer than 15 nonzero digits: the computed determinant is rounded at the appropriate digit position to restore some or all of the accuracy lost to floating-point arithmetic.

This refinement technique can cause the computed determinant to exhibit discontinuity. To avoid this, you can disable the refinement by setting flag -54.

**Example:** For a square matrix \( A \), the minor of element \( a_{ij} \) is the determinant of the submatrix that remains after deleting row \( i \) and column \( j \) from the original matrix. Given a square matrix in level 3, \( i \) in level 2, and \( j \) in level 1, the following program, MINOR determines the minor of the submatrix:

```
« + M row col
  « M row ROW- DROP col COL- DROP DET
  »
»
```

For example, entering \([[1 2 3] [4 5 6] [7 8 9]] 2 3 MINOR\) returns -6.

**Related Commands:** CNRM, CROSS, DOT, RNRM
DETACH

Detach Library Command: Detaches the library with the specified number from the current directory. Each library has a unique number. If a port number is specified, it is ignored.

- \( n_{\text{library}} \rightarrow \)
- \( n_{\text{port}} : n_{\text{library}} \rightarrow \)

Keyboard Access: \( \leftarrow \text{LIBRARY} \) DETACH

Affected by Flags: None

Remarks: A RAM-based library object attached to the HOME directory must be detached before it can be purged, whereas a library attached to any other directory does not. Also, a library object attached to a non-HOME directory is automatically detached (without using DETACH) whenever a new library object is attached there.

Related Commands: ATTACH, LIBS, PURGE

DIAG

Array to Matrix Diagonal Command: Takes an array and a specified dimension and returns a matrix whose major diagonal elements are the elements of the array.

- \([\text{array}]_{\text{diagonals}} \rightarrow \{\dim\} \rightarrow [[\text{matrix}]]\)

Keyboard Access: \( \text{MTH} \text{MATR} \text{NXT} \text{DIAG} \)

Affected by Flags: None
Remarks: Real number dimensions are rounded to integers. If a single dimension is given, a square matrix is returned. If two dimensions are given, the proper order is \{ number of rows, number of columns \}. No more than two dimensions can be specified.

If the main diagonal of the resulting matrix has more elements than the array, additional diagonal elements are set to zero. If the main diagonal of the resulting matrix has fewer elements than the array, extra array elements are dropped.

Related Commands: →DIAG

---

→DIAG

Matrix Diagonal to Array Command: Returns a vector that contains the major diagonal elements of a matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[ matrix ]]</td>
<td>→</td>
<td>[ vector ]_diagonals</td>
</tr>
</tbody>
</table>

Keyboard Access: \[ \textup{MTH} \, \textup{MATR} \, \textup{NXT} \, \rightarrow \text{DIAG} \]

Affected by Flags: None

Remarks: The input matrix does not have to be square.

Related Commands: DIAG→
DIFFEQ

Differential Equation Plot Type Command: Sets the plot type to DIFFEQ.

Keyboard Access: («)(PLOT) TYPE DIFFEQ

Affected by Flags: None

Remarks: When the plot type is DIFFEQ and the reserved variable EQ does not contain a list, the initial value problem is solved and plotted over an interval using the Runge-Kutta-Fehlberg (4,5) method. The plotting parameters are specified in the reserved variable PPAR, which has the following form:

\[
\langle (x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}}) \text{ indep res axes ptype depend} \rangle
\]

For plot type DIFFEQ, the elements of PPAR are used as follows:

- \( (x_{\text{min}}, y_{\text{min}}) \) is a complex number specifying the lower left corner of PICT (the lower left corner of the display range). The default value is \((-6.5, -3.1)\).

- \( (x_{\text{max}}, y_{\text{max}}) \) is a complex number specifying the upper right corner of PICT (the upper right corner of the display range). The default value is \((6.5, 3.2)\).

- \text{indep} is a list, \( \{ 'X', x_0, x_f \} \), containing a name that specifies the independent variable, and two numbers that specify the initial and final values for the independent variable. The default values for these elements are \( \{ 'X', \emptyset, x_{\text{max}} \} \).

- \text{res} is a real number specifying the maximum interval, in user-unit coordinates, between values of the independent variable. The default value is \( \emptyset \). If \text{res} does not equal zero, then the maximum interval is \text{res}. If \text{res} equals zero, the maximum interval is unlimited.

- \text{axes} is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. If the solution is real-valued, these strings can specify the dependent or the independent variable; if the solution is vector valued, the strings can specify a solution component:
  - "0" specifies the independent variable (X)
"1" specifies the dependent variable ($Y$)

"n" specifies a solution component $Y_n$

If $axes$ contains any strings other than "0", "1", or "n", the DIFFEQ-plotter uses the default strings "0" and "1", and plots the independent variable on the horizontal axis and the dependent variable on the vertical.

- $ptype$ is a command name specifying the plot type. Executing the command DIFFEQ places the command name DIFFEQ in $PPAR$.

- $depend$ is a list, $\{'Y' \; y_0 \; x_{ErrTol}\}$, containing a name that specifies the dependent variable (the solution), and two numbers that specify the initial value of $Y$ and the global absolute error tolerance in the solution $Y$. The default values for these elements are $\{'Y' \; 0 \; 0.0001\}$.

$EQ$ must define the right-hand side of the initial value problem $Y'(X)=F(X,Y)$. $Y$ can return a real value or real vector when evaluated.

The DIFFEQ-plotter attempts to make the interval between values of the independent variable as large as possible, while keeping the computed solution within the specified error tolerance $x_{ErrTol}$. This tolerance may hold only at the computed points. Straight lines are drawn between computed step endpoints, and these lines may not accurately represent the actual shape of the solution. $res$ limits the maximum interval size to provide higher plot resolution.

On exit from DIFFEQ plot, the first elements of $indep$ and $depend$ (identifiers) contain the final values of $X$ and $Y$, respectively.

If $EQ$ contains a list, the initial value problem is solved and plotted using a combination of Rosenbrock (3,4) and Runge-Kutta-Fehlberg (4,5) methods. In this case DIFFEQ uses RRKSTEP to calculate $y_j$, and $EQ$ must contain two additional elements:

- The second element of $EQ$ must evaluate to the partial derivative of $Y'$ with respect to $X$, and can return a real value or real vector when evaluated.

- The third element of $EQ$ must evaluate to the partial derivative of $Y'$ with respect to $Y$, and can return a real value or a real matrix when evaluated.
**DISP**

**Display Command:** Displays \( obj \) in the \( n \)th display line.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( obj )</td>
<td>( n )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG][NXT] [OUT] DISP

**Affected by Flags:** None

**Remarks:** \( n \leq 1 \) indicates the top line of the display; \( n \geq 7 \) indicates the bottom line.

To facilitate the display of messages, strings are displayed without the surrounding " " delimiters. All other objects are displayed in the same form as would be used if the object were in level 1 in the multiline display format. If the object display requires more than one display line, the display starts in line \( n \), and continues down the display either to the end of the object or the bottom of the display.

The object displayed by DISP persists in the display only until the keyboard is ready for input. The FREEZE command can be used to cause the object to persist in the display until a key is pressed.

**Example:** The program

```
"ENTER Data Now" 1 DISP 7 FREEZE HALT
```

displays ENTER Data Now at the top of the display, "freezes" the entire display, and halts.

**Related Commands:** FREEZE, HALT, INPUT, PROMPT
**DO Indefinite Loop Structure Command:** Starts DO ... UNTIL ... END indefinite loop structure.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DO</strong></td>
<td>→</td>
<td></td>
</tr>
<tr>
<td><strong>UNTIL</strong></td>
<td>→</td>
<td></td>
</tr>
<tr>
<td><strong>END</strong></td>
<td><strong>T/F</strong></td>
<td>→</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] [BRCH] £ [DO] £ [DO] £

**Affected by Flags:** None

**Remarks:** DO ... UNTIL ... END executes a loop repeatedly until a test returns a true (nonzero) result. Since the test clause is executed after the loop clause, the loop is always executed at least once. The syntax is:

\[
\text{DO } \text{loop-clause} \text{ UNTIL } \text{test-clause} \text{ END}
\]

DO starts execution of the loop clause. UNTIL ends the loop clause and begins the test clause. The test clause must return a test result to the stack. END removes the test result from the stack. If its value is zero, the loop clause is executed again; otherwise, execution resumes following END.

**Example:** The following program counts down from 100 to 0 and leaves the integers 100 to 0 on the stack:

```
«
100 'A' STO A
DO 'A' DECR
UNTIL 'A==0'
END
'A' PURGE
»
```
DO

Related Commands: END, UNTIL, WHILE

DOERR

Do Error Command: Executes a "user-specified" error, causing a program to behave exactly as if a normal error had occurred during program execution.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{nerror}</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>\text{#nerror}</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>&quot;error&quot;</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \text{PRG} \text{NXT} \text{ERROR DOERR}

Affected by Flags: None

Remarks: DOERR causes a program to behave exactly as if a normal error has occurred during program execution. The error message depends on the argument provided to DOERR:

- \text{nerror} or \text{#nerror} display the corresponding built-in error message.
- "error" displays the contents of the string. (A subsequent execution of ERRM returns "error". ERRN returns \# 70000h.)
- \text{∅} abandons program execution without displaying a message—∅ DOERR is equivalent to pressing \text{CANCEL}.

Example: The following program takes a number from the stack and returns an error if the number is greater than 10:

```
\text{« \text{×}}
\text{CASE}
\text{'\text{x} > 10'}
\text{THEN "\text{x} IS TOO BIG" DOERR END}
```
**DOLIST**

*Do to List Command:* Applies commands, programs, or user-defined functions to lists.

**Related Commands:** ERRM, ERRN, ERRO

---

### Level n...Level 3  | Level 2  | Level 1  | → | Level 1
--- | --- | --- | --- | ---
{ list }₁ ... { list }ₙ | n | program | → | { results }
{ list }₁ ... { list }ₙ | n | command | → | { results }
{ list }₁ ... { list }ₙ | n | name | → | { results }
  { list }₁ ... { list }ₙ | program | → | { results }
  { list }₁ ... { list }ₙ | command | → | { results }
  { list }₁ ... { list }ₙ | name | → | { results }

---

**Keyboard Access:** PRG LIST PROC DOLIS

**Affected by Flags:** None

**Remarks:** The number of lists, n, can be omitted when the level 1 argument is any of the following:

- A command.
- A program containing exactly one command (e.g. \( \langle \text{DUP} \rangle \)).
- A program conforming to the structure of a user-defined function.

The level 1 object can be a local or global name that refers to a program or command.

All lists must be the same length l. The program is executed l times: on the ith iteration, n objects each taken from the ith position in each list are entered on the stack in the same order as in their original lists, and the program is executed. The results from each execution are left on the stack. After the final iteration, any new results are combined into a single list.
DOLIST

Example:  \( (1 2 3)(4 5 6)(7 8 9)3 \times + \times \) DOLIST returns \( (11 26 45) \).

Related Commands: DOSUBS, ENDSUB, NSUB, STREAM

DOSUBS

Do to Sublist Command: Applies a program or command to groups of elements in a list.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ list }_1</td>
<td>( n )</td>
<td>( \ll program \rr )</td>
<td>→</td>
<td>{ list }_2</td>
</tr>
<tr>
<td>{ list }_1</td>
<td>( n )</td>
<td>command</td>
<td>→</td>
<td>{ list }_2</td>
</tr>
<tr>
<td>{ list }_1</td>
<td>( n )</td>
<td>name</td>
<td>→</td>
<td>{ list }_2</td>
</tr>
<tr>
<td>{ list }_1</td>
<td>( \ll program \rr )</td>
<td>→</td>
<td>{ list }_2</td>
<td></td>
</tr>
<tr>
<td>{ list }_1</td>
<td>command</td>
<td>→</td>
<td>{ list }_2</td>
<td></td>
</tr>
<tr>
<td>{ list }_1</td>
<td>name</td>
<td>→</td>
<td>{ list }_2</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: [PRG] LIST PROC DOSUB

Affected by Flags: None

Remarks: The real number \( n \) can be omitted when the level 1 argument is one of the following:

- A command.
- A user program containing a single command.
- A program with a user-defined function structure.
- A global or local name that refers to one of the above.

The first iteration uses elements 1 through \( n \) from the list; the second iteration uses elements 2 through \( n+1 \); and so on. In general, the \( m^{th} \) iteration uses the elements from the list corresponding to positions \( m \) through \( m+n-1 \).

During an iteration, the position of the first element used in that iteration is available to the user using the command NSUB, and the number of groups of elements is available using the command...
ENDSUB. Both of these commands return an Undefined Local Name error if executed when DOSUBS is not active.

DOSUBS returns the Invalid User Function error if the level 1 argument is a user program that does not contain only one command and does not have a user-defined function structure. DOSUBS also returns the Wrong Argument Count error if the level 1 argument is a command that does not accept 1 to 5 arguments of specific types (DUP, ROT, or LIST, for example).

Examples: \(\{A B C D E\} \leftarrow \Rightarrow \) DOSUBS returns \(\{A-B, B-C, C-D, D-E\}\).

\(\{A B C\} 2 \leftarrow \Rightarrow \) DOSUBS returns \(\{A*(B*B), B*(C*C)\}\).

Entering
\[
\{1 2 3 4 5\} \rightarrow a \ b
\]
\(\langle \text{CASE 'NSUB==1' THEN a END} '\text{NSUB==ENDSUB' THEN b END} 'a+b' \text{EVAL} \rangle \text{EVAL} \rangle \text{EVAL}
\(\rangle \text{EVAL} \rangle \text{EVAL}
\) DOSUBS returns \(\{1 5 7 5\}\).

Related Commands: DOLIST, ENDSUB, NSUB, STREAM
**DOT**

**Dot Product Command:** Returns the dot product \( A \cdot B \) of two arrays \( A \) and \( B \), calculated as the sum of the products of the corresponding elements of the two arrays.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[array A]</td>
<td>[array B]</td>
<td>( \rightarrow )</td>
<td>( x )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH} \ \text{VECTR} \ \text{DOT} \)

**Affected by Flags:** None

**Remarks:** Both arrays must have the same dimensions.

Some authorities define the dot product of two complex arrays as the sum of the products of the conjugated elements of one array with their corresponding elements from the other array. The HP 48 uses the ordinary products without conjugation. If you prefer the alternate definition, apply CONJ to one or both arrays before using DOT.

**Example:** \([1 \ 2 \ 3] \ [4 \ 5 \ 6] \ \text{DOT} \) returns 32 (by calculating \( 1 \times 4 + 2 \times 5 + 3 \times 6 \)).

**Related Commands:** CNRM, CROSS, DET, RNRM

---

**DRAW**

**Draw Plot Command:** Plots the mathematical data in the reserved variable \( EQ \) or the statistical data in the reserved variable \( \Sigma DAT \), using the specified \( x \)- and \( y \)-axis display ranges.

**Keyboard Access:** \( \text{\textasciitilde} \text{PLOT} \ \text{DRAW} \)

**Affected by Flags:** Simultaneous or Sequential Plot (–28), Curve Filling (–31)

**Remarks:** The plot type determines if the data in the reserved variable \( EQ \) or the data in the reserved variable \( \Sigma DAT \) is plotted.
DRAW does not erase $PICT$ before plotting—execute ERASE to do so. DRAW does not draw axes—execute DRAX to do so.

When DRAW is executed from a program, the graphics display, which shows the resultant plot, does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

**Related Commands:** AUTO, AXES, DRAX, ERASE, FREEZE, PICTURE, LABEL, PVIEW

---

**DRAX**

**Draw Axes Command:** Draws axes in $PICT$.

**Keyboard Access:** $(e)(PLOT)$ DRAX

**Affected by Flags:** None

**Remarks:** The coordinates of the axes intersection are specified by AXES. Axes tick-marks are specified in PPAR with the ATICK, or AXES command. DRAX does not draw axes labels—execute LABEL to do so.

**Related Commands:** AXES, DRAW, LABEL

---

**DROP**

**Drop Object Command:** Removes the level 1 object from the stack.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$obj$</td>
<td>$\rightarrow$</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** $(e)DROP$

**Affected by Flags:** None

**Related Commands:** CLEAR, DROPN, DROP2
**DROPN**

**Drop n Objects Command**  Removes the first \( n + 1 \) objects from the stack (the first \( n \) objects excluding the integer \( n \) itself).

<table>
<thead>
<tr>
<th>Level ( n+1 ) ... Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( obj_1 ) ... ( obj_n )</td>
<td>( n )</td>
<td>( \rightarrow )</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:**  \( \leftarrow \text{STACK} \text{ NXT} \text{ DRPN} \)

**Affected by Flags:**  None

**Related Commands:**  CLEAR, DROP, DROP2

---

**DROP2**

**Drop 2 Objects Command:**  Removes the first two objects from the stack.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( obj_1 )</td>
<td>( obj_2 )</td>
<td>( \rightarrow )</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:**  \( \leftarrow \text{STACK} \text{ NXT} \text{ DRPO2} \)

**Affected by Flags:**  None

**Related Commands:**  CLEAR, DROP, DROPN
**DTAG**

**Delete Tag Command:** DTAG removes all tags (labels) from an object.

| Level 1       | → | Level 1
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>:tag:obj</td>
<td>→</td>
<td>obj</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] TYPE [NXT] DTAG

**Affected by Flags:** None

**Remarks:** The leading colon is not shown for readability when the tagged object is on the stack.

DTAG has no effect on an untagged object.

**Related Commands:** LIST→, →TAG

---

**DUP**

**Duplicate Object Command:** DUP returns a copy to level 1 of the object in level 1.

| Level 1       | → | Level 2 | Level 1
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>→</td>
<td>obj</td>
<td>obj</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

Pressing [ENTER] duplicates the item on level 1.

[STACK] [NXT] DUP

**Affected by Flags:** None

**Related Commands:** DUPN, DUP2, PICK

Command Reference 3-97
**DUPN**

**Duplicate n Objects Command:** Takes an integer $n$ from level 1 of the stack, and returns copies of the objects in stack levels $2n$ through $n + 1$.

<table>
<thead>
<tr>
<th>Lvl $n+1$...Lvl 2</th>
<th>Lvl 1</th>
<th>$\rightarrow$</th>
<th>Lvl $2n$...Lvl $n+1$</th>
<th>Lvl $n$...Lvl 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$obj_n...obj_1$</td>
<td>$n$</td>
<td>$\rightarrow$</td>
<td>$obj_n...obj_1$</td>
<td>$obj_n...obj_1$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{(«)(STACK)(NXT)(DUPN)} \)

**Affected by Flags:** None

**Related Commands:** DUP, DUP2, PICK

---

**DUP2**

**Duplicate 2 Objects Command:** DUP2 returns copies of the objects in levels 1 and 2 of the stack.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$obj_2$</td>
<td>$obj_1$</td>
<td>$\rightarrow$</td>
<td>$obj_2$</td>
<td>$obj_1$</td>
<td>$obj_2$</td>
<td>$obj_1$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{(«)(STACK)(NXT)(DUP2)} \)

**Affected by Flags:** None

**Related Commands:** DUP, DUPN, PICK
**D→R**

**Degrees to Radians Function:** Converts a real number representing an angle in degrees to its equivalent in radians.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>→</td>
<td>$(\pi/180)\ x$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'D→R(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [REAL] [NXT] [NXT] [D→R]

**Affected by Flags:** Numerical Results (−3)

**Remarks:** This function operates independently of the angle mode.

**Related Commands** R→D

---

**e**

**e Function:** Returns the symbolic constant $e$ or its numerical representation, 2.71828182846.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>'e'</td>
</tr>
<tr>
<td></td>
<td>→</td>
<td>2.71828182846</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- [MTH] [NXT] [CONS] \( e \)

**Affected by Flags:** Symbolic Constants (−2), Numerical Results (−3)
When evaluated, \( e \) returns its numerical representation if flag \(-2\) or flag \(-3\) is set; otherwise, \( e \) returns its symbolic representation.

**Remarks:** The number returned for \( e \) is the closest approximation of the constant \( e \) to 12-digit accuracy. For exponentiation, use the expression \( \exp(x) \) rather than \( e^x \), since the function \( \exp \) uses a special algorithm to compute the exponential to greater accuracy.

**Related Commands:** \( \exp, \expm, i, \ln, \lnp1, \maxr, \minr, \pi \)

---

**EGV**

**Eigenvalues and Eigenvectors Command:** Computes the eigenvalues and right eigenvectors for a square matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
</table>
| \( \text{
\[ \text{matrix} \text{]}_A \] \} \) | \rightarrow | \( \text{
\[ \text{matrix} \text{]}_E\text{Vec} \] \) | \( \text{[vector]}_E\text{Val} \) |

**Keyboard Access:** MTH MATR NXT EGV

**Affected by Flags:** None

**Remarks:** The resulting vector \( E\text{Val} \) contains the computed eigenvalues. The columns of matrix \( E\text{Vec} \) contain the right eigenvectors corresponding to the elements of vector \( E\text{Val} \).

The computed results should minimize (within computational precision):

\[
\frac{|A \cdot E\text{Vec} - E\text{Vec} \cdot \text{diag}(E\text{Val})|}{n \cdot |A|}
\]

where \( \text{diag}(E\text{Val}) \) denotes the \( n \times n \) diagonal matrix containing the eigenvalues \( E\text{Val} \).

**Related Commands:** EGVL

---

3-100 Command Reference
**EGVL**

**Eigenvalues Command:** Computes the eigenvalues of a square matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\text{matrix}]_A$</td>
<td>→</td>
<td>$[\text{vector}]_{\text{Eval}}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH} \ \text{MATR} \ \text{NXT} \ \text{EGVL} \)

**Affected by Flags:** None

**Remarks:** The resulting vector \( \text{L} \) contains the computed eigenvalues.

**Related Commands:** EGV

---

**ELSE**

**ELSE Command:** Starts false clause in conditional or error-trapping structure.

See the IF and IFERR keyword entries for syntax information.

**Keyboard Access:**

- \( \text{PRG} \ \text{BRCH} \ \text{IF} \ \text{ELSE} \)
- \( \text{PRG} \ \text{NXT} \ \text{ERROR} \ \text{IFERR} \ \text{ELSE} \)

**Remarks:** See the IF and IFERR keyword entries for more information.

**Related Commands:** IF, IFERR, THEN, END
END

**END Command:** Ends conditional, error-trapping, and indefinite loop structures.

See the IF, CASE, IFERR, DO, and WHILE keyword entries for syntax information.

**Keyboard Access:**

```
PRG  BRCH  IF  END
PRG  BRCH  CASE  END
PRG  BRCH  DO  END
PRG  BRCH  WHILE  END
PRG  NXT  ERROR  IFERR  END
```

**Remarks:** See the IF, CASE, IFERR, DO, and WHILE keyword entries for more information.

**Related Commands:** IF, CASE, DO, ELSE, IFERR, REPEAT, THEN, UNTIL, WHILE,

ENDSUB

**Ending Sublist Command:** Provides a way to access the total number of sublists contained in the list used by DOSUBS.

**Keyboard Access:**

```
PRG  LIST  PROC  ENDS
```

**Affected by Flags:** None

**Remarks:** Returns an Undefined Local Name error if executed when DOSUBS is not active.

**Example:** The following program subtracts the number of elements in a list from each element in the list:

```
« + a
« a 1
« ENDSUB -
```
Related Commands: DOSUBS, NSUB

**Engineering Mode Command:** Sets the number display format to Engineering mode, which displays one to three digits to the left of the fraction mark (decimal point) and an exponent that is a multiple of three. The total number of significant digits displayed is \( n + 1 \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MODES} \) \( \text{FMT} \) \( \text{ENG} \)

**Affected by Flags:** None

**Remarks:** Engineering mode uses \( n + 1 \) significant digits, where \( 0 \leq n \leq 11 \). (Values for \( n \) outside this range are rounded up or down.) A number is displayed or printed as follows:

\[
(\text{sign}) \text{ mantissa E (sign) exponent}
\]

where the mantissa is of the form \((nn)n.(n \ldots )\) (with up to 12 digits total) and the exponent has one to three digits.

A number with an exponent of \(-499\) is displayed automatically in Scientific mode.

**Example:** The number 103.6 in Engineering mode with five significant digits \((n=4)\) would appear as \(103.600\text{E}0\). This same number with one significant digit \((n=0)\) would appear as \(100.\text{E}0\).

**Related Commands:** FIX, SCI, STD
**EQ**

**Equation to Stack Command:** Separates an equation into its left and right sides.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb₁ = symb₂'</td>
<td>→</td>
<td>'symb₁'</td>
<td>→</td>
<td>'symb₂'</td>
</tr>
<tr>
<td>z</td>
<td>→</td>
<td>z</td>
<td>→</td>
<td>0</td>
</tr>
<tr>
<td>'name'</td>
<td>→</td>
<td>'name'</td>
<td>→</td>
<td>0</td>
</tr>
<tr>
<td>x_unit</td>
<td>→</td>
<td>x_unit</td>
<td>→</td>
<td>0</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'symb'</td>
<td>→</td>
<td>0</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \[\text{PRG TYPE NXT EQ=}\]

**Affected by Flags:** None

**Remarks:** If the argument is an expression, it is treated as an equation whose right side equals zero.

**Related Commands:** ARRY→, DTAG, LIST→, OBJ→, STR→

**EQNLIB**

**Equation Library Command:** Starts the Equation Library application.

**Keyboard Access:** \[\text{EQ LIB EQLIB EQNLIB}\]

**Affected by Flags:** None

**Remarks:** The Equation Library is a collection of equations and commands useful for solving typical science and engineering problems.

**Related Commands:** MSOLVR, SOLVEQN
ERASE

Erase PICT Command: Erases PICT, leaving a blank PICT of the same dimensions.

Keyboard Access: 
- PICTURE EDIT NXT ERASE
- PICTURE CLEAR
- PLOT ERASE

Affected by Flags: None

Related Commands: DRAW

ERRM

Error Message Command: Returns a string containing the error message of the most recent calculator error.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>&quot;error message&quot;</td>
</tr>
</tbody>
</table>

Keyboard Access: PRG NXT ERROR ERRM

Affected by Flags: None

Remarks: ERRM returns the string for an error generated by DOERR. If the argument to DOERR was ∅, the string returned by ERRM is empty.

Example: The program « IFERR + THEN ERRM END » returns "Bad Argument Type" to level 1 if improper arguments (for example, a complex number and a binary integer) are in levels 1 and 2.

Related Commands: DOERR, ERRN, ERR0
ERRN

**Error Number Command:** Returns the error number of the most recent calculator error.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>#error</td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG NXT ERROR ERRN

**Affected by Flags:** None

**Remarks:** If the most recent error was generated by DOERR with a string argument, ERRN returns # 70000h. If the most recent error was generated by DOERR with a binary integer argument, ERRN returns that binary integer. (If the most recent error was generated by DOERR with a real number argument, ERRN returns the binary integer conversion of the real number.)

**Example:** The program « IFERR + THEN ERRN END » returns # 202h to level 1 if improper arguments (for example, a complex number and a binary integer) are in levels 1 and 2.

**Related Commands:** DOERR, ERRM, ERR0

ERR0

**Clear Last Error Number Command:** Clears the last error number so that a subsequent execution of ERRN returns # 0h, and clears the last error message.

**Keyboard Access:** PRG NXT ERROR ERR0

**Affected by Flags:** None

**Related Commands:** DOERR, ERRM, ERRN
Evaluate Object Command: Evaluates the object.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( obj )</td>
<td>( \rightarrow )</td>
<td>(see below)</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{EVAL} \)

Affected by Flags: Numerical Results \((-3)\)

Remarks: The following table describes the effect of the evaluation on different object types.

<table>
<thead>
<tr>
<th>Obj. Type</th>
<th>Effect of Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Name</td>
<td>Recalls the contents of the variable.</td>
</tr>
<tr>
<td>Global Name</td>
<td>\textit{Calls} the contents of the variable:</td>
</tr>
<tr>
<td></td>
<td>- A name is evaluated.</td>
</tr>
<tr>
<td></td>
<td>- A program is evaluated.</td>
</tr>
<tr>
<td></td>
<td>- A directory becomes the current directory.</td>
</tr>
<tr>
<td></td>
<td>- Other objects are put on the stack.</td>
</tr>
<tr>
<td></td>
<td>If no variable exists for a given name, evaluating the name returns the name to the stack.</td>
</tr>
<tr>
<td>Program</td>
<td>\textit{Enters} each object in the program:</td>
</tr>
<tr>
<td></td>
<td>- Names are evaluated (unless quoted).</td>
</tr>
<tr>
<td></td>
<td>- Commands are evaluated.</td>
</tr>
<tr>
<td></td>
<td>- Other objects are put on the stack.</td>
</tr>
<tr>
<td>List</td>
<td>\textit{Enters} each object in the list:</td>
</tr>
<tr>
<td></td>
<td>- Names are evaluated.</td>
</tr>
<tr>
<td></td>
<td>- Commands are evaluated.</td>
</tr>
<tr>
<td></td>
<td>- Programs are evaluated.</td>
</tr>
<tr>
<td></td>
<td>- Other objects are put on the stack.</td>
</tr>
</tbody>
</table>
**EVAL**

<table>
<thead>
<tr>
<th>Obj. Type</th>
<th>Effect of Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged</td>
<td>If the tag specifies a port, recalls and evaluates the specified object. Otherwise, puts the untagged object on the stack.</td>
</tr>
<tr>
<td>Algebraic</td>
<td><em>Enters</em> each object in the algebraic expression:</td>
</tr>
<tr>
<td></td>
<td>- Names are evaluated.</td>
</tr>
<tr>
<td></td>
<td>- Commands are evaluated.</td>
</tr>
<tr>
<td></td>
<td>- Other objects are put on the stack.</td>
</tr>
<tr>
<td>Command, Function, XLIB Name</td>
<td>Evaluates the specified object.</td>
</tr>
<tr>
<td>Other Objects</td>
<td>Puts the object on the stack.</td>
</tr>
</tbody>
</table>

To evaluate a symbolic argument to a numerical result, evaluate the argument in Numerical Result mode (flag -3 set) or execute →NUM on that function.

**Related Commands:** →NUM, SYSEVAL

**EXP**

**Exponential Analytic Function:** Returns the exponential, or natural antilogarithm, of the argument; that is, $e$ raised to the given power.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>→</td>
<td>$e^z$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'EXP(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (2) (2)

**Affected by Flags:** Numerical Results (-3)

3-108 Command Reference
**Remarks:** EXP uses extended precision constants and a special algorithm to compute its result to full 12-digit precision for all arguments that do not trigger and underflow or overflow error.

EXP provides a more accurate result for the exponential than can be obtained by using \( e^z \). The difference in accuracy increases as \( z \) increases. For example:

<table>
<thead>
<tr>
<th>( z )</th>
<th>EXP(( z ))</th>
<th>( e^z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>20.0855369232</td>
<td>20.0855369232</td>
</tr>
<tr>
<td>10</td>
<td>22026.4657948</td>
<td>22026.4657949</td>
</tr>
<tr>
<td>100</td>
<td>2.68811714182E43</td>
<td>2.68811714191E43</td>
</tr>
<tr>
<td>500</td>
<td>1.40359221785E217</td>
<td>1.40359221809E217</td>
</tr>
<tr>
<td>1000</td>
<td>1.97007111402E434</td>
<td>1.97007111469E434</td>
</tr>
</tbody>
</table>

For complex arguments,
\[
e^{(x,y)} = e^x \cos y + ie^x \sin y
\]

**Related Commands:** ALOG, EXPM, LN, LOG

---

**EXPAN**

**Expand Products Command:** Rewrites an algebraic expression or equation by expanding products and powers.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb(_1)'</td>
<td>→</td>
<td>'symb(_2)'</td>
</tr>
<tr>
<td>( x )</td>
<td>→</td>
<td>( x )</td>
</tr>
<tr>
<td>( (x, y) )</td>
<td>→</td>
<td>( (x, y) )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** ←(SYMBOLIC) EXPAN

**Affected by Flags:** None

**Examples:** 'A*(B+C)' EXPAN returns 'A*B+A*C'.

Command Reference 3-109
EXPAN

'\(A^{(B+C)}\) EXPAN returns 'A^B*A^C'.

'\(X^5\) EXPAN returns 'X*X*X*X*X'.

'(X+Y)^2' EXPAN returns 'X^2+2*X*Y+Y^2'.

**Related Commands:** COLCT, ISOL, QUAD, SHOW

EXPFIT

**Exponential Curve Fit Command:** Stores EXPFIT as the fifth parameter in the reserved variable \(\Sigma PAR\), indicating that subsequent executions of LR are to use the exponential curve fitting model.

**Keyboard Access:** («)(STAT) 2PAR MODL EXPFIT

**Affected by Flags:** None

**Remarks:** LINFIT is the default specification in \(\Sigma PAR\). For a description of \(\Sigma PAR\), see appendix D, "Reserved Variables."

**Related Commands:** BESTFIT, LR, LINFIT, LOGFIT, PWRFIT

EXPM

**Exponential Minus 1 Analytic Function:** Returns \(e^x - 1\).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>→</td>
<td>(e^x - 1)</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'EXPM(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [HYP] (NXT) EXPM

**Affected by Flags:** Numerical Results (-3)

**Remarks:** For values of \(x\) close to zero, '\(\text{EXPM}(x)\)' returns a more accurate result than does '\(\exp(x)-1\)'. (Using EXPM allows
both the argument and the result to be near zero, and avoids an intermediate result near 1. The calculator can express numbers within $10^{-449}$ of zero, but within only $10^{-11}$ of 1.)

**Related Commands:** EXP, LNP1

---

**EYEPT**

**Eye Point Command:** Specifies the coordinates of the eye point in a perspective plot.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1 →</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{\text{point}}$</td>
<td>$y_{\text{point}}$</td>
<td>$z_{\text{point}}$</td>
<td>→</td>
</tr>
</tbody>
</table>

**Keyboard Access:** ←(PLOT) NXT 3D VPAR NXT EYEPT

**Affected by Flags:** None

**Remarks:** $x_{\text{point}}$, $y_{\text{point}}$, and $z_{\text{point}}$ are real numbers that set the x-, y-, and z-coordinates as the eye-point from which to view a 3D plot’s view volume. The y-coordinate must always be 1 unit less than the view volume’s nearest point ($y_{\text{near}}$ of YVOL). These coordinates are stored in the reserved variable VPAR.

**Related Commands:** NUMX, NUMY, XVOL, XXRNG, YVOL, YYRNG, ZVOL
**F0λ**

**Black Body Emissive Power Function:** Returns the fraction of total black-body emissive power.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_{\text{lambda}} )</td>
<td>( x_T )</td>
<td>→</td>
<td>( x_{\text{power}} )</td>
</tr>
<tr>
<td>( y_{\text{lambda}} )</td>
<td>'symb'</td>
<td>→</td>
<td>'F0λ(( y_{\text{lambda}} ),\text{symb})'</td>
</tr>
<tr>
<td>'symb'</td>
<td>( x_T )</td>
<td>→</td>
<td>'F0λ(\text{symb},x_T)'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>'symb_2'</td>
<td>→</td>
<td>'F0λ(\text{symb}_1,\text{symb}_2)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

**Affected by Flags:** Numerical Results (-3)

**Remarks:** F0λ calculates the fraction of total black-body emissive power at temperature \( x_T \) between wavelengths 0 and \( y_{\text{lambda}} \). If units are not specified, \( y_{\text{lambda}} \) has implied units of meters and \( x_T \) has implied units of K.

F0λ returns a dimensionless fraction.

---

**FACT**

**Factorial (Gamma) Function:** Provided for compatibility with the HP 28. FACT is the same as !. See !.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>→</td>
<td>( n! )</td>
</tr>
<tr>
<td>( x )</td>
<td>→</td>
<td>( \Gamma(x+1) )</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'(symb)!'</td>
</tr>
</tbody>
</table>
FANNING

Fanning Friction Factor Function: Calculates the Fanning friction factor of certain fluid flows.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{x_D}{D} )</td>
<td>( y_{Re} )</td>
<td>→</td>
<td>( x_{fanning} )</td>
</tr>
<tr>
<td>( \frac{x_D}{D} )</td>
<td>'symb'</td>
<td>→</td>
<td>'FANNING(( x_D/D ), symb)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>( y_{Re} )</td>
<td>→</td>
<td>'FANNING(symb, ( y_{Re} ))'</td>
</tr>
<tr>
<td>'symb1'</td>
<td>'symb2'</td>
<td>→</td>
<td>'FANNING(symb1, symb2)'</td>
</tr>
</tbody>
</table>

Keyboard Access: None. Must be typed in.

Affected by Flags: Numerical Results (−3)

Remarks: FANNING calculates the Fanning friction factor, a correction factor for the frictional effects of fluid flows having constant temperature, cross-section, velocity, and viscosity (a typical pipe flow, for example). \( \frac{x_D}{D} \) is the relative roughness (the ratio of the conduit roughness to its diameter). \( y_{Re} \) is the Reynolds number. The function uses different computation routines for laminar flow (\( Re \leq 2100 \)) and turbulent flow (\( Re > 2100 \)). \( \frac{x_D}{D} \) and \( y_{Re} \) must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Related Commands: DARCY
**FC?**

**Flag Clear? Command:** Tests whether the system or user flag specified by \( n_{\text{flag number}} \) is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{\text{flag number}} )</td>
<td>→</td>
<td>0/1</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

```
PRG TEST NXT NXT FC?
<MODES FLAG FC?
```

**Affected by Flags:** None

**Related Commands:** CF, FC?C, FS?, FS?C, SF

---

**FC?C**

**Flag Clear? Clear Command:** Tests whether the system or user flag specified by \( n_{\text{flag number}} \) is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set. After testing, clears the flag.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{\text{flag number}} )</td>
<td>→</td>
<td>0/1</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

```
PRG TEST NXT NXT FC?C
<MODES FLAG FC?C
```

3-114  Command Reference
**FFT**

**Affected by Flags:** None

**Example:** If flag -44 is set, -44 FC?C returns 0 to level 1 and clears flag -44.

**Related Commands:** CF, FC?, FS?, FS?C, SF

---

**FFT**

**Discrete Fourier Transform Command:** Computes the one- or two-dimensional discrete Fourier transform of an array.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ array ]_1</td>
<td>→</td>
<td>[ array ]_2</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [NXT] **FFT** **FFT**

**Affected by Flags:** None

**Remarks:** If the argument is an $N$-vector or an $N \times 1$ or $1 \times N$ matrix, FFT computes the one-dimensional transform. If the argument is an $M \times N$ matrix, FFT computes the two-dimensional transform. $M$ and $N$ must be integral powers of 2.

The one-dimensional discrete Fourier transform of an $N$-vector $X$ is the $N$-vector $Y$ where:

$$Y_k = \sum_{n=0}^{N-1} X_n e^{-\frac{2\pi i kn}{N}}, \quad i = \sqrt{-1}$$

for $k = 0, 1, \ldots, N-1$.

The two-dimensional discrete Fourier transform of an $M \times N$ matrix $X$ is the $M \times N$ matrix $Y$ where:

$$Y_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{mn} e^{-\frac{2\pi i km}{M}} e^{-\frac{2\pi i ln}{N}}, \quad i = \sqrt{-1}$$

for $k = 0, 1, \ldots, M-1$ and $l = 0, 1, \ldots, N-1$. 

Command Reference 3-115
FFT

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The FFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

Related Commands: IFFT

FINDALARM

Find Alarm Command: Returns the alarm index \( n_{index} \) of the first alarm due after the specified time.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>→</td>
<td>( n_{index} )</td>
</tr>
<tr>
<td>{ date time }</td>
<td>→</td>
<td>( n_{index} )</td>
</tr>
<tr>
<td>0</td>
<td>→</td>
<td>( n_{index} )</td>
</tr>
</tbody>
</table>

Keyboard Access: \(<\text{TIME}>\text{ALRM}\text{FINDA}\)

Affected by Flags: Date Format (-42)

Remarks: If the level 1 argument is a real number date, FINDALARM returns the index of the first alarm due after 12:00 AM on that date. If the argument is a list \( \{ \text{date time} \} \), it returns the index of the first alarm due after that date and time. If the argument is the real number 0, FINDALARM returns the first past-due alarm.

For any of the three arguments, FINDALARM returns 0 if no alarm is found.

Related Commands: DELALARM, RCLALARM, STOALARM
FINISH

Finish Server Mode Command: Terminates Kermit Server mode in a device connected to an HP 48

Keyboard Access: \[\text{SRCR VR FINIS}\]

Affected by Flags: I/O Device flag (-33), I/O Messages (-39)

Remarks: FINISH is used by a local Kermit device to tell a server Kermit (connected via the serial port or the IR port) to exit Server mode.

Related Commands: BAUD, CKSM, KGET, PARITY, PKT, RECN, RECV, SEND, SERVER

FIX

Fix Mode Command: Sets the number display format to Fix mode, which rounds the display to \(n\) decimal places.

\[
\begin{array}{c|c}
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
n & \rightarrow & \\
\end{array}
\]

Keyboard Access: \[\text{SRCR VR FMT FIX}\]

Affected by Flags: None

Remarks: Fix mode shows \(n\) digits to the right of the fraction mark (decimal point), where \(0 \leq n \leq 11\). (Values for \(n\) outside this range are rounded to the nearest integer.) A number is displayed or printed as follows:

\[(\text{sign}) \text{ mantissa}\]

where the mantissa can be of any form. However, the calculator automatically displays a number in Scientific mode if either of the following is true:
**FIX**

- The number of digits to be displayed exceeds 12.
- A nonzero value rounded to \( n \) decimal places otherwise would be displayed as zero.

**Example:** The number 103.6 in Fix mode to four decimal places would appear as 103.6000.

**Related Commands:** FIX, SCI, STD

---

**FLOOR**

**Floor Function:** Returns the greatest integer that is less than or equal to the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>→</td>
<td>( n )</td>
</tr>
<tr>
<td>( x_{\text{unit}} )</td>
<td>→</td>
<td>( n_{\text{unit}} )</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'FLOOR(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:**  
[MTH](#) REAL ![NXT](#) ![NXT](#) FLOOR

**Affected by Flags:** Numerical Results \((-3)\)

**Examples:** 3.2 \( \text{FLOOR} \) returns 3; \(-3.2\) \( \text{FLOOR} \) returns -4.

**Related Commands:** CEIL, IP, RND, TRNC
FOR

FOR Definite Loop Structure Command: Starts FOR ... NEXT and FOR ... STEP definite loop structures.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR</td>
<td>$x_{\text{start}}$</td>
<td>$x_{\text{finish}}$</td>
<td>→</td>
</tr>
<tr>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOR</td>
<td>$x_{\text{start}}$</td>
<td>$x_{\text{finish}}$</td>
<td>→</td>
</tr>
<tr>
<td>STEP</td>
<td></td>
<td>$x_{\text{increment}}$</td>
<td>→</td>
</tr>
<tr>
<td>STEP</td>
<td></td>
<td>'symb_{increment}'</td>
<td>→</td>
</tr>
</tbody>
</table>

Keyboard Access:
To begin a definite loop:

PRG BRCH FOR FOR

To type FOR NEXT

PRG BRCH ← FOR

To type FOR STEP:

PRG BRCH → FOR

Affected by Flags: None

Remarks: Definite loop structures execute a command or sequence of commands a specified number of times.

- A FOR ... NEXT loop executes a program segment a specified number of times using a local variable as the loop counter. You can use this variable within the loop. The syntax is this:

$$x_{\text{start}} \ x_{\text{finish}} \ \text{FOR} \ \text{counter} \ \text{loop-clause} \ \text{NEXT}$$

FOR takes $x_{\text{start}}$ and $x_{\text{finish}}$ from the stack as the beginning and ending values for the loop counter, then creates the local variable counter as a loop counter. Then, the loop clause is executed; counter can be referenced or have its value changed within the loop clause. NEXT increments counter by one, and then tests whether
counter is less than or equal to \( x_{\text{finish}} \). If so, the loop clause is repeated (with the new value of counter).

When the loop is exited, counter is purged.

FOR \( \ldots \) STEP works just like FOR \( \ldots \) NEXT, except that it lets you specify an increment value other than 1. The syntax is:

\[
x_{\text{start}} \ x_{\text{finish}} \ \text{FOR} \ \text{counter loop-clause} \ x_{\text{increment}} \ \text{STEP}
\]

FOR takes \( x_{\text{start}} \) and \( x_{\text{finish}} \) from the stack as the beginning and ending values for the loop counter, then creates the local variable counter as a loop counter. Next, the loop clause is executed; counter can be referenced or have its value changed within the loop clause. STEP takes \( x_{\text{increment}} \) from the stack and increments counter by that value. If the argument of STEP is an algebraic expression or a name, it is automatically evaluated to a number.

The increment value can be positive or negative. If the increment is positive, the loop is executed again when counter is less than or equal to \( x_{\text{finish}} \). If the increment is negative, the loop is executed when counter is greater than or equal to \( x_{\text{finish}} \).

When the loop is exited, counter is purged.

**Example:** The following program sums all odd integers in the range 1 to 100:

\[
\ll 0 \ 1 \ 100 \\
\text{FOR I I + 2 STEP}
\]

**Related Commands:** NEXT, START, STEP
**FP**

**Fractional Part Function:** Returns the fractional part of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>→</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_{\text{unit}}$</td>
<td>→</td>
<td>$y_{\text{unit}}$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'FP(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH REAL NXT] FP

**Affected by Flags:** Numerical Results (−3)

**Remarks:** The result has the same sign as the argument.

**Examples:** -32.3 FP returns −.3; 32.3 FP returns .3.

**Related Commands:** IP

---

**FREE**

**Free RAM Card Command:** Frees (makes independent) the previously merged RAM in port 1. FREE is provided for compatibility with the HP 48SX, which could merge RAM in port 2 as well. See FREE1.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td></td>
<td>→</td>
<td>$n_{\text{port}}$</td>
</tr>
<tr>
<td>${\text{name}<em>{\text{backup}} \ldots \text{n}</em>{\text{library}}}$</td>
<td></td>
<td>→</td>
<td>$n_{\text{port}}$</td>
</tr>
<tr>
<td>name_{\text{backup}}</td>
<td></td>
<td>→</td>
<td>$n_{\text{port}}$</td>
</tr>
<tr>
<td>$n_{\text{library}}$</td>
<td></td>
<td>→</td>
<td>$n_{\text{port}}$</td>
</tr>
</tbody>
</table>
**FREE**

**Keyboard Access:** None. Must be typed in.

**Affected by Flags:** None

**Remarks:** Any prior contents of the port are moved into user memory. If level 2 specifies backup or library objects, those objects are moved from port 0 to the newly freed RAM port.

**Related Commands:** FREE1, MERGE1

---

**FREE1**

**Free RAM Card Command:** Frees (makes independent) the previously merged RAM in port 1. Any prior contents of the port are moved into user memory. If level 1 specifies backup or library objects, those objects are moved from port 0 to the newly freed RAM port.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{name_{backup} \cdots n_{library}}</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>name_{backup}</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>n_{library}</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\leftarrow\text{LIBRARY}\) FREE1

**Affected by Flags:** None

**Remarks:** The list in level 1 can be empty (in which case no objects are moved to the newly independent RAM) or it can contain any number of backup names and library numbers. Level 1 cannot be completely empty, however.

**Related Commands:** FREE, MERGE, MERGE1
**FREEZE**

**FREEZE Display Command:** Freezes the part of the display specified by \( n_{\text{display area}} \), so that it is not updated until a key is pressed.

<table>
<thead>
<tr>
<th>Level 1 ( \rightarrow ) Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{\text{display area}} ) ( \rightarrow )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \[ \text{PRG} \, \text{NXT} \, \text{OUT} \, \text{FREEZ} \]

**Affected by Flags:** None

**Remarks:** Normally, the stack display is updated as soon as the calculator is ready for data input. For example, when HALT stops a running program, or when a program ends, any displayed messages are cleared. The FREEZE command “freezes” a part or all of the display so that it is not updated until a key is pressed. This allows, for example, a prompting message to persist after a program halts to await data input.

\( n_{\text{display area}} \) is the sum of the value codes for the areas to be frozen:

<table>
<thead>
<tr>
<th>Display Area</th>
<th>Value Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status area</td>
<td>1</td>
</tr>
<tr>
<td>Stack/Command-line area</td>
<td>2</td>
</tr>
<tr>
<td>Menu area</td>
<td>4</td>
</tr>
</tbody>
</table>

So, for example, 2 FREEZE freezes the stack/command-line area, 3 FREEZE freezes the status area and the stack/command-line area, and 7 FREEZE freezes all three areas.

Values of \( n_{\text{display area}} \geq 7 \) or \( \leq 0 \) freeze the entire display (are equivalent to value 7). To freeze the graphics display, you must freeze the status and stack/command-line areas (by entering 3), or the entire display (by entering 7).
**FREEZE**

**Examples:**

This program:

```
"Ready for data" 1 DISP 1 FREEZE HALT
```

displays the contents of the string in the top line of the display, then freezes the status area so that the string contents persist in the display after HALT is executed:

This program:

```
{ # 0d # 0d } PVView 7 FREEZE
```

selects the graphics display and then freezes the entire display so that the graphics display persists after the program ends. (If FREEZE was not executed, the stack display would be selected after the program ends.)

To use FREEZE with PVView (or any graphics display), you must enter 3 or 7.

**Related Commands:** CLLCD, DISP, HALT

---

**FS?**

**Flag Set? Command:** Tests whether the system or user flag specified by `nflag` is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nflag</code></td>
<td>→</td>
<td>0/1</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- `PRG` `TEST` `NXT` `NXT` `FS?`
- `AZY` `MODES` `FLAG` `FS?`

**Affected by Flags:** None

**Related Commands:** CF, FC?, FC?C, FS?C, SF

3-124  Command Reference
**FUNCTION**

**FS?C**

**Flag Set? Clear Command:** Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear. After testing, clears the flag.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$n_{\text{flag number}}$</th>
<th>Level 1</th>
<th>0/1</th>
</tr>
</thead>
</table>

**Keyboard Access:**

- `PRG TEST NXT NXT FS?C`
- `MODES FLAG FS?C`

**Affected by Flags:** None

**Example:** If flag -44 is set, -44 FS?C returns 1 to level 1 and clears flag -44.

**Related Commands:** CF, FC?, FC?C, FS?, SF

---

**FUNCTION**

**Function Plot Type Command:** Sets the plot type to FUNCTION.

**Keyboard Access:**

- `(PLOT) PTYPE FUNC`

**Affected by Flags:** Simultaneous Plotting (-28), Curve Filling (-31)

**Remarks:** When the plot type is FUNCTION, the DRAW command plots the current equation as a real-valued function of one real variable. The current equation is specified in the reserved variable `EQ`. The plotting parameters are specified in the reserved variable `PPAR`, which has the form:

```
{ (x_min, y_min) (x_max, y_max) indep res axes ptype depend }
```
For plot type FUNCTION, the elements of PPAR are used as follows:

- \( (x_{\text{min}}, y_{\text{min}}) \) is a complex number specifying the lower left corner of PICT (the lower left corner of the display range). The default value is \( (-6.5, -3.1) \).

- \( (x_{\text{max}}, y_{\text{max}}) \) is a complex number specifying the upper right corner of PICT (the upper right corner of the display range). The default value is \( (6.5, 3.2) \).

- \( \text{indep} \) is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of \( \text{indep} \) is \( X \).

- \( \text{res} \) is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is \( 0 \), which specifies an interval of 1 pixel.

- \( \text{axes} \) is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is \( (0, 0) \).

- \( \text{ptype} \) is a command name specifying the plot type. Executing the command FUNCTION places the name FUNCTION in PPAR.

- \( \text{depend} \) is a name specifying a label for the vertical axis. The default value is \( Y \).

The current equation is plotted as a function of the variable specified in \( \text{indep} \). The minimum and maximum values of the independent variable (the plotting range) can be specified in \( \text{indep} \); otherwise, the values in \( (x_{\text{min}}, y_{\text{min}}) \) and \( (x_{\text{max}}, y_{\text{max}}) \) (the display range) are used. Lines are drawn between plotted points unless flag -31 is set.

If \( EQ \) contains an expression or program, the expression or program is evaluated in Numerical Results mode for each value of the independent variable to give the values of the dependent variable. If \( EQ \) contains an equation, the plotting action depends on the form of the equation, as shown in the following table.
### Form of Current Equation

<table>
<thead>
<tr>
<th>Form</th>
<th>Plotting Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'expr=expr'</code></td>
<td>Each expression is plotted separately. The intersection of the two graphs shows where the expressions are equal.</td>
</tr>
<tr>
<td><code>'name=expr'</code></td>
<td>Only the expression is plotted.</td>
</tr>
<tr>
<td><code>'indep=constant'</code></td>
<td>A vertical line is plotted.</td>
</tr>
</tbody>
</table>

If flag -28 is set, all equations are plotted simultaneously.

If the independent variable in the current equation represents a unit object, you must specify the units by storing a unit object in the corresponding variable in the current directory. For example, if the current equation is `'X+3_m'`, and you want `X` to represent some number of inches, you would store `1_in` (the number part of the unit object is ignored) in `X`. For each plotted point, the numerical value of the independent variable is combined with the specified unit (inches in this example) before the current equation is evaluated. If the result is a unit object, only the number part is plotted.

**Related Commands:** BAR, CONIC, DIFFEQ, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

---

**GET**

**Get Element Command:** Returns from the level 2 array or list (or named array or list) the real or complex number `z_{get}` or object `obj_{get}` whose position is specified in level 1.
## GET

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[ matrix ]]</td>
<td>(n_{\text{position}})</td>
<td>→</td>
<td>(z_{\text{get}})</td>
</tr>
<tr>
<td>[[ matrix ]]</td>
<td>({n_{\text{row}}\ \ m_{\text{col}}})</td>
<td>→</td>
<td>(z_{\text{get}})</td>
</tr>
<tr>
<td>'name_matrix'</td>
<td>(n_{\text{position}})</td>
<td>→</td>
<td>(z_{\text{get}})</td>
</tr>
<tr>
<td>'name_matrix'</td>
<td>({n_{\text{row}}\ \ m_{\text{col}}})</td>
<td>→</td>
<td>(z_{\text{get}})</td>
</tr>
<tr>
<td>[ vector ]</td>
<td>(n_{\text{position}})</td>
<td>→</td>
<td>(z_{\text{get}})</td>
</tr>
<tr>
<td>[ vector ]</td>
<td>({n_{\text{position}}})</td>
<td>→</td>
<td>(z_{\text{get}})</td>
</tr>
<tr>
<td>'name_vector'</td>
<td>(n_{\text{position}})</td>
<td>→</td>
<td>(z_{\text{get}})</td>
</tr>
<tr>
<td>'name_vector'</td>
<td>({n_{\text{position}}})</td>
<td>→</td>
<td>(z_{\text{get}})</td>
</tr>
<tr>
<td>{ list }</td>
<td>(n_{\text{position}})</td>
<td>→</td>
<td>(\text{obj}_{\text{get}})</td>
</tr>
<tr>
<td>{ list }</td>
<td>({n_{\text{position}}})</td>
<td>→</td>
<td>(\text{obj}_{\text{get}})</td>
</tr>
<tr>
<td>'name_list'</td>
<td>(n_{\text{position}})</td>
<td>→</td>
<td>(\text{obj}_{\text{get}})</td>
</tr>
<tr>
<td>'name_list'</td>
<td>({n_{\text{position}}})</td>
<td>→</td>
<td>(\text{obj}_{\text{get}})</td>
</tr>
</tbody>
</table>

### Keyboard Access:
- **PRG**
- **LIST**
- **ELEM**
- **GET**

### Affected by Flags:
None

### Remarks:
For matrices, \(n_{\text{position}}\) is incremented in row order.

### Examples:
- \[[ 2 \ 3 \ 7 \ 1 \ 9 ]\ [ 2 \ 1 \ 3 ]\] \((2\ 3)\) GET returns 9.
- \[[ 2 \ 3 \ 7 \ 1 \ 9 \ 2 \ 1 \ 3 \ 1 ]\] GET returns 1.
- \((A \ B \ C \ D \ E)\ \{1 \ 3\}\) GET returns 'A'.

### Related Commands:
- GETI, PUT, PUTI
### GETI

**Get and Increment Index Command:** Returns from the level 2 array or list (or named array or list) the real or complex number $z_{get}$ or object $obj_{get}$ whose position is specified in level 1, along with the level 2 argument and the next position in that argument.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[matrix]]</td>
<td>$n_{pos1}$</td>
<td>—</td>
<td>[[matrix]]</td>
<td>$n_{pos2}$</td>
</tr>
<tr>
<td>[[matrix]]</td>
<td>{ $n_r$ $m_c$ }$</td>
<td>—</td>
<td>[[matrix]]</td>
<td>{ $n_r$ $m_c$ }$_2$</td>
</tr>
<tr>
<td>'name$_{mtrx}'</td>
<td>$n_{pos1}$</td>
<td>—</td>
<td>'name$_{mtrx}'</td>
<td>$n_{pos2}$</td>
</tr>
<tr>
<td>'name$_{mtrx}'</td>
<td>{ $n_r$ $m_c$ }$</td>
<td>—</td>
<td>'name$_{mtrx}'</td>
<td>{ $n_r$ $m_c$ }$_2$</td>
</tr>
<tr>
<td>[ vec ]</td>
<td>$n_{pos1}$</td>
<td>—</td>
<td>[ vec ]</td>
<td>$n_{pos2}$</td>
</tr>
<tr>
<td>[ vec ]</td>
<td>{ $n_{pos1}$ }</td>
<td>—</td>
<td>[ vec ]</td>
<td>{ $n_{pos2}$ }</td>
</tr>
<tr>
<td>'name$_{vec}$'</td>
<td>$n_{pos1}$</td>
<td>—</td>
<td>'name$_{vec}$'</td>
<td>$n_{pos2}$</td>
</tr>
<tr>
<td>'name$_{vec}$'</td>
<td>{ $n_{pos1}$ }</td>
<td>—</td>
<td>'name$_{vec}$'</td>
<td>{ $n_{pos2}$ }</td>
</tr>
<tr>
<td>{ list }</td>
<td>$n_{pos1}$</td>
<td>—</td>
<td>{ list }</td>
<td>$n_{pos2}$</td>
</tr>
<tr>
<td>{ list }</td>
<td>{ $n_{pos1}$ }</td>
<td>—</td>
<td>{ list }</td>
<td>{ $n_{pos2}$ }</td>
</tr>
<tr>
<td>'name$_{list}$'</td>
<td>$n_{pos1}$</td>
<td>—</td>
<td>'name$_{list}$'</td>
<td>$n_{pos2}$</td>
</tr>
<tr>
<td>'name$_{list}$'</td>
<td>{ $n_{pos1}$ }</td>
<td>—</td>
<td>'name$_{list}$'</td>
<td>{ $n_{pos2}$ }</td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG LIST ELEM GETI

**Affected by Flags:** Index Wrap Indicator (-64)

The Index Wrap Indicator flag is cleared on each execution of GETI until the position index wraps to the first position in the argument, at which point the flag is set. The next execution of GETI clears the flag again.

**Remarks:** For matrices, the position is incremented in row order.

**Related Commands:** GET, PUT, PUTI
GOR

Graphics OR Command: Superimposes \( grob_1 \) onto \( grob_{target} \) or \( PICT \), with the upper left corner pixel of \( grob_1 \) positioned at the specified coordinate in \( grob_{target} \) or \( PICT \).

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( grob_{target} )</td>
<td>{ #n, #m }</td>
<td>( grob_1 )</td>
<td>→</td>
<td>( grob_{result} )</td>
</tr>
<tr>
<td>( grob_{target} )</td>
<td>( (x,y) )</td>
<td>( grob_1 )</td>
<td>→</td>
<td>( grob_{result} )</td>
</tr>
<tr>
<td>( PICT )</td>
<td>{ #n, #m }</td>
<td>( grob_1 )</td>
<td>→</td>
<td>( grob_{result} )</td>
</tr>
<tr>
<td>( PICT )</td>
<td>( (x,y) )</td>
<td>( grob_1 )</td>
<td>→</td>
<td>( grob_{result} )</td>
</tr>
</tbody>
</table>

Keyboard Access: [PRG] GROB GOR

Affected by Flags: None

Remarks: GOR uses a logical OR to determine the state (on or off) of each pixel in the overlapping portion of the argument graphics object.

If the level 3 argument is any graphics object other than \( PICT \), then \( grob_{result} \) is returned to the stack. If the level 3 argument is \( PICT \), no result is returned to the stack.

Any portion of \( grob_1 \) that extends past \( grob_{target} \) or \( PICT \) is truncated.

Related Commands: GXOR, REPL, SUB
**GRAD**

**Grads Mode Command:** Sets Grads angle mode.

**Keyboard Access:** \( \leftarrow \text{MODES} \) \text{ANGL} \text{GRAD} \)

**Affected by Flags:** None

**Remarks:** GRAD clears flag —17 and sets flag —18, and displays the GRAD annunciator.

In Grads angle mode, real-number arguments that represent angles are interpreted as grads, and real-number results that represent angles are expressed in grads.

**Related Commands:** DEG, RAD

---

**GRAPH**

**Picture Environment Command:** Selects the Picture environment (selects the graphics display and activates the graphics cursor and Picture menu).

**Keyboard Access:** None. Must be typed in.

**Affected by Flags:** None

**Remarks:** GRAPH is provided for compatibility with the HP 48S series, and is the same as the PICTURE command. See PICTURE.

**Related Commands:** PICTURE, PVIEW, TEXT
GRIDMAP

GRIDMAP Plot Type Command: Sets plot type to GRIDMAP.

Keyboard Access: \begin{center} \texttt{PLOT NXT 3D RTYPE GRID} \end{center}

Affected by Flags: None

Remarks: When plot type is set GRIDMAP, the DRAW command plots a mapping grid representation of a 2-vector-valued function of two variables. GRIDMAP requires values in the reserved variables \textit{EQ}, \textit{VPAR}, and \textit{PPAR}.

\textit{VPAR} has the following form:

\begin{align*}
\{ x_{\text{left}}, x_{\text{right}}, y_{\text{near}}, y_{\text{far}}, z_{\text{low}}, z_{\text{high}}, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}}, x_{\text{step}}, y_{\text{step}} \}.
\end{align*}

For plot type GRIDMAP, the elements of \textit{VPAR} are used as follows:

- \textit{x_{left}} and \textit{x_{right}} are real numbers that specify the width of the view space.
- \textit{y_{near}} and \textit{y_{far}} are real numbers that specify the depth of the view space.
- \textit{z_{low}} and \textit{z_{high}} are real numbers that specify the height of the view space.
- \textit{x_{min}} and \textit{x_{max}} are real numbers the specify the input region’s width. The default value is \((-1, 1)\).
- \textit{y_{min}} and \textit{y_{max}} are real numbers that specify the input region’s depth. The default value is \((-1, 1)\).
- \textit{x_{eye}}, \textit{y_{eye}}, and \textit{z_{eye}} are real numbers that specify the point in space from which you view the graph.
- \textit{x_{step}} and \textit{y_{step}} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted. These can be used instead of (or in combination with) \textit{RES}.

The plotting parameters are specified in the reserved variable \textit{PPAR}, which has the following form:

\begin{align*}
\langle x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}}, \text{indep}, \text{res}, \text{axes}, \text{ptype}, \text{depend} \rangle
\end{align*}

For plot type GRIDMAP, the elements of \textit{PPAR} are used as follows:

- \langle x_{\text{min}}, y_{\text{min}} \rangle is not used.
- \((x_{\text{max}}, y_{\text{max}})\) is not used.

- \textit{indep} is a name specifying the independent variable. The default value of \textit{indep} is \(X\).

- \textit{res} is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.

- \textit{axes} is not used.

- \textit{ptype} is a command name specifying the plot type. Executing the command GRIDMAP places the command name GRIDMAP in \textit{PPAR}.

- \textit{depend} is a name specifying the dependent variable. The default value is \(Y\).

- **Related Commands:** BAR, CONIC, DIFFEQ, FUNCTION, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

---

**Stack to Graphics Object Command:** Creates a graphics object representing the level 2 object, where the argument \(n_{\text{char size}}\) specifies the character size of the representation.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>(\rightarrow)</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{obj}</td>
<td>(n_{\text{char size}})</td>
<td>(\rightarrow)</td>
<td>\textit{grob}</td>
</tr>
</tbody>
</table>

- **Keyboard Access:** \(\text{PRG GROB GROB}\)

- **Affected by Flags:** None

- **Remarks:** \(n_{\text{char size}}\) can be 0, 1 (small), 2 (medium), or 3 (large). \(n_{\text{char size}} = 0\) is the same as \(n_{\text{char size}} = 3\), except for unit objects
and algebraic objects, where 0 specifies the EquationWriter application picture.

**Example:** This program:

```
« 'Y=3*X^2' 0 →GROB PICT STO { } PVIEW »
```

returns a graphics object to the stack representing the EquationWriter application picture of 'Y=3*X^2', then stores the graphics object in `PICT` and shows it in the graphics display with scrolling activated.

**Related Commands:** →LCD, LCD→

---

**GXOR**

**Graphics Exclusive OR Command:** Superimposes `grob_1` onto `grob_target` or `PICT`, with the upper left corner pixel of `grob_1` positioned at the specified coordinate in `grob_target` or `PICT`.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>grob_target</code></td>
<td><code>{ #n #m }</code></td>
<td><code>grob_1</code></td>
<td>→</td>
<td><code>grob_result</code></td>
</tr>
<tr>
<td><code>grob_target</code></td>
<td><code>(x,y)</code></td>
<td><code>grob_1</code></td>
<td>→</td>
<td><code>grob_result</code></td>
</tr>
<tr>
<td><code>PICT</code></td>
<td><code>{ #n #m }</code></td>
<td><code>grob_1</code></td>
<td>→</td>
<td><code>grob_result</code></td>
</tr>
<tr>
<td><code>PICT</code></td>
<td><code>(x,y)</code></td>
<td><code>grob_1</code></td>
<td>→</td>
<td><code>grob_result</code></td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG GROB GXOR

**Affected by Flags:** None

**Remarks:** GXOR is used for creating cursors, for example, to make the cursor image appear dark on a light background and light on a dark background. Executing GXOR again with the same image restores the original picture.

GXOR uses a logical exclusive OR to determine the state of the pixels (on or off) in the overlapping portion of the argument graphics objects.
Any portion of grob that extends past grob\textsubscript{target} or PICT is truncated.

If the level 3 argument (the target graphics object) is any graphics object other than PICT, then grob\textsubscript{result} is returned to the stack. If the level 3 argument is PICT, no result is returned to the stack.

**Example:** This program:

```
000000
!GEOERLASTARG
```

turns on (makes dark) every pixel in PICT, then superimposes a 5 \times 5 graphics object on PICT at pixel coordinates \{ #0d #0d \}. Each on-pixel in the 5 by 5 graphics object turns off (makes light) the corresponding pixel in PICT. Then, the original picture is restored by executing GXOR again with the same arguments.

**Related Commands:** GOR, REPL, SUB

---

**Multiply Height Command:** Multiplies the vertical plot scale by \( x_{\text{factor}} \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{\text{factor}} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{PLOT} \) \( \text{PPAR} \) \( \text{NXT} \) \( \ast \text{H} \)

**Affected by Flags:** None

**Remarks:** Executing \( \ast \text{H} \) changes the \( y \)-axis display range—the \( y_{\text{min}} \) and \( y_{\text{max}} \) components of the first two complex numbers in the reserved variable \( \text{PPAR} \). The plot origin (the user-unit coordinate of the center pixel) is not changed.

**Related Commands:** AUTO, \( \ast \text{W} \), YRNG
HALT

Halt Program Command: Halts program execution.

Keyboard Access: [PRG] [NXT] [RUN] [HALT]

Affected by Flags: None

Remarks: Program execution is halted at the location of the HALT command in the program. The HALT annunciator is turned on. Program execution is resumed by executing CONT (usually by pressing [PRG] [NXT] [RUN] [CONT]). Executing KILL (usually by pressing [PRG] [NXT] [RUN] [KILL]) cancels all halted programs.

Related Commands: CONT, KILL

HEAD

First Listed Element Command: Returns the first element of a list or string.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ obj₁ ... objₙ }</td>
<td>→</td>
<td>obj₁</td>
</tr>
<tr>
<td>&quot;string&quot;</td>
<td>→</td>
<td>&quot;element₁&quot;</td>
</tr>
</tbody>
</table>

Keyboard Access:

[PRG] [LIST] [ELEM] [NXT] [HEAD]

[CHARS] [NXT] [HEAD]

Affected by Flags: None

Example: "Dead" HEAD returns "D".

The following program takes a list of coordinates { A B C } that define a right triangle, and finds the length of the hypotenuse AC:

```
SWAP REVLIST HEAD ABS
```

For example, entering `{ (0,0) (0,3) (3,4) }` returns 5.
HEX

Hexadecimal Mode Command: Selects hexadecimal base for binary integer operations. (The default base is decimal.)

Keyboard Access: [MTH] [BASE] [HEX]

Affected by Flags: Binary Integer Words size (-5 through -10), Binary Integer Base (-11, -12)

Remarks: Binary integers require the prefix #. Binary integers entered and returned in hexadecimal base automatically show the suffix h. If the current base is not hexadecimal, then you can enter a hexadecimal number by ending it with h. It will be displayed in the current base when it is entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Related Commands: BIN, DEC, OCT, RCWS, STWS

HISTOGRAM

Histogram Plot Type Command: Sets the plot type to HISTOGRAM.

Keyboard Access: [PLOT] [NXT] [STAT] [PTYPE] [HISTO]

Affected by Flags: None

Remarks: When the plot type is HISTOGRAM, the DRAW command creates a histogram using data from one column of the current statistics matrix (reserved variable \( \Sigma DAT \)). The column is specified by the first parameter in the reserved variable \( \Sigma PAR \) (using the XCOL command). The plotting parameters are specified in the reserved variable \( PPAR \), which has the form:

\[
\langle x_{min}, y_{min} \rangle \langle x_{max}, y_{max} \rangle \text{ indep res axes ptype depend } \rangle
\]
HISTOGRAM

For plot type HISTOGRAM, the elements of PPAR are used as follows:

- \( (x_{\text{min}}, y_{\text{min}}) \) is a complex number specifying the lower left corner of PICT (the lower left corner of the display range). The default value is \((-6.5, -3.5)\).

- \( (x_{\text{max}}, y_{\text{max}}) \) is a complex number specifying the upper right corner of PICT (the upper right corner of the display range). The default value is \((6.5, 3.5)\).

- indep is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers that specify the minimum and maximum values of the data to be plotted. The default value of indep is \(X\).

- res is a real number specifying the bin size, in user-unit coordinates, or a binary integer specifying the bin size in pixels. The default value is \(0\), which specifies the bin size to be \(1/13\) of the difference between the specified minimum and maximum values of the data.

- axes is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is \((0, 0)\).

- ptype is a command name specifying the plot type. Executing the command HISTOGRAM places the command name HISTOGRAM in PPAR.

- depend is a name specifying a label for the vertical axis. The default value is \(Y\).

The frequency of the data is plotted as bars, where each bar represents a collection of data points. The base of each bar spans the values of the data points, and the height indicates the number of data points. The width of each bar is specified by res. The overall maximum and minimum values for the data can be specified by indep; otherwise, the values in \((x_{\text{min}}, y_{\text{min}})\) and \((x_{\text{max}}, y_{\text{max}})\) are used.

Related Commands: BAR, CONIC, DIFEQ, FUNCTION, GRIDMAP, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE
HISTPLOT

Draw Histogram Plot Command: Plots a frequency histogram of the specified column in the current statistics matrix (reserved variable \( \Sigma \text{DAT} \)).

Keyboard Access: \( \leftrightarrow \text{STAT} \) PLOT HISTP

Affected by Flags: None

Remarks: The data column to be plotted is specified by XCOL and is stored as the first parameter in the reserved variable \( \Sigma \text{PAR} \). If no data column is specified, column 1 is selected by default. The y-axis is autoscaled and the plot type is set to HISTOGRAM.

HISTPLOT plots relative frequencies, using 13 bins as the default number of partitions. The RES command lets you specify a different number of bins by specifying the bin width. To plot a frequency histogram with numerical frequencies, store the frequencies in \( \Sigma \text{DAT} \) and execute BINS and then BARPLOT.

When HISTPLOT is executed from a program, the graphics display, which shows the resultant plot, does not persist unless PICTURE, PVVIEW (with an empty list argument), or FREEZE is subsequently executed.

Related Commands: BARPLOT, BINS, FREEZE, PICTURE, PVVIEW, RES, SCATRPLOT, XCOL

HMS+

Hours-Minutes-Seconds Plus Command: Returns the sum of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{HMS}_1 )</td>
<td>( \text{HMS}_2 )</td>
<td>→</td>
<td>( \text{HMS}_1 + \text{HMS}_2 )</td>
</tr>
</tbody>
</table>
HMS+

Keyboard Access: \(< TIME >\) \(\text{NXT} \) \(\text{HMS+}\)

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is \(H.MMSSs\), where:
- \(H\) is zero or more digits representing the integer part of the number.
- \(MM\) are two digits representing the number of minutes.
- \(SS\) are two digits representing the number of seconds.
- \(s\) is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: \(\text{HMS→}\), \(→\text{HMS}\), \(\text{HMS−}\)

---

HMS−

Hours-Minutes-Seconds Minus Command: Returns the difference of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(HMS_1)</td>
<td>(HMS_2)</td>
<td>→</td>
<td>(HMS_1−HMS_2)</td>
</tr>
</tbody>
</table>

Keyboard Access: \(< TIME >\) \(\text{NXT} \) \(\text{HMS−}\)

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is \(H.MMSSs\), where:
- \(H\) is zero or more digits representing the integer part of the number.
- \(MM\) are two digits representing the number of minutes.
- \(SS\) are two digits representing the number of seconds.
- \(s\) is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.
**HMS→**

**Hours-Minutes-Seconds to Decimal Command:** Converts a real number in hours-minutes-seconds format to its decimal form (hours or degrees with a decimal fraction).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMS</td>
<td>→</td>
<td>x</td>
</tr>
</tbody>
</table>

**Keyboard Access:**  

\[ ← \text{TIME} \text{NXT} \text{HMS}+ \]

**Affected by Flags:** None

**Remarks:** The format for HMS (a time or an angle) is \( H.MMSSs \), where:

- \( H \) is zero or more digits representing the integer part of the number.
- \( MM \) are two digits representing the number of minutes.
- \( SS \) are two digits representing the number of seconds.
- \( s \) is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

**Related Commands:** →HMS, HMS+, HMS−
Decimal to Hours-Minutes-Seconds Command: Converts a real number representing hours or degrees with a decimal fraction to hours-minutes-seconds format.

### Level 1 \( \rightarrow \) Level 1

| \( x \) | \( \rightarrow \) | \( HMS \) |

**Keyboard Access:** \( (\uparrow) \) TIME \( \text{NXT} \) \( \text{HMS}\rightarrow \)

**Affected by Flags:** None

**Remarks:** The format for HMS (a time or an angle) is \( H.MMSSs \), where:

- \( H \) is zero or more digits representing the integer part of the number.
- \( MM \) are two digits representing the number of minutes.
- \( SS \) are two digits representing the number of seconds.
- \( s \) is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

**Related Commands:** \( \text{HMS}\rightarrow, \text{HMS}+, \text{HMS}− \)

---

**HOME**

**HOME Directory Command:** Makes the \( HOME \) directory the current directory.

**Keyboard Access:** \( (\uparrow) \) HOME

**Affected by Flags:** None

**Related Commands:** CRDIR, PATH, PGDIR, UPDIR
i

i Function: Returns the symbolic constant $i$ or its numerical representation, $(0, 1)$.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\rightarrow$</td>
<td>'i'</td>
</tr>
<tr>
<td></td>
<td>$\rightarrow$</td>
<td>$(0,1)$</td>
</tr>
</tbody>
</table>

Keyboard Access:

- $\alpha \leftrightarrow I$
- $\text{MTH} \ \text{NXT} \ \text{CONS} \ I$

Affected by Flags: Symbolic Constants $(-2)$, Numerical Results $(-3)$

Evaluating $i$ returns its numerical representation if flag $-2$ or $-3$ is set; otherwise, its symbolic representation is returned.

Related Commands: $e$, MAXR, MINR, $\pi$

IDN

Identity Matrix Command: Returns an identity matrix; that is, a square matrix with its diagonal elements equal to 1 and its off-diagonal elements equal to 0.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$\rightarrow$</td>
<td>[[ R-matrix identity ]]</td>
</tr>
<tr>
<td>[[ matrix ]]</td>
<td>$\rightarrow$</td>
<td>[[ matrix identity ]]</td>
</tr>
<tr>
<td>'name'</td>
<td>$\rightarrow$</td>
<td></td>
</tr>
</tbody>
</table>
### IDN

**Keyboard Access:** [MTH] [MATR] [MAKE] [IDN]

**Affected by Flags:** None

**Remarks:** The result is either a new square matrix, or it’s an existing square matrix with its elements replaced by the elements of the identity matrix, according to the argument in level 1.

- **Creating a new matrix:** If the argument is a real number $n$, a new real identity matrix is returned to level 1, with its number of rows and number of columns equal to $n$.

- **Replacing the elements of an existing matrix:** If the argument is a square matrix, an identity matrix of the same dimensions is returned. If the original matrix is complex, the resulting identity matrix will also be complex, with diagonal values $(1, 0)$.

  If the argument is a name, the name must identify a variable containing a square matrix. In this case, the elements of the matrix are replaced by those of the identity matrix (complex if the original matrix is complex).

**Related Commands:** CON

### IF

**IF Conditional Structure Command:** Starts IF ... THEN ... END and IF ... THEN ... ELSE ... END conditional structures.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>THEN</td>
<td>T/F</td>
<td>→</td>
</tr>
<tr>
<td>END</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>THEN</td>
<td>T/F</td>
<td>→</td>
</tr>
<tr>
<td>ELSE</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>
IF

Keyboard Access:  

Affected by Flags:  None

Remarks: Conditional structures, used in combination with program tests, enable a program to make decisions.

IF ... THEN ... END executes a sequence of commands only if a test returns a nonzero (true) result. The syntax is:

\[
\text{IF } \text{test-clause} \text{ THEN } \text{true-clause} \text{ END}
\]

IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is nonzero, the true clause is executed. Otherwise, program execution resumes following END.

IF ... THEN ... ELSE ... END executes one sequence of commands if a test returns a true (nonzero) result, or another sequence of commands if that test returns a false (zero) result. The syntax is:

\[
\text{IF } \text{test-clause} \text{ THEN } \text{true-clause} \text{ ELSE } \text{false-clause} \text{ END}
\]

IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is nonzero, the true clause is executed. Otherwise, the false clause is executed. After the appropriate clause is executed, execution resumes following END.

The test clause can be a command sequence (for example, \( A \geq B \)) or an algebraic (for example, \( \frac{1}{2} \times (\text{SIN} (A)) \)). If the test clause is an algebraic, it is automatically evaluated to a number (→NUM or EVAL isn’t necessary).

Related Commands: CASE, ELSE, END, IFERR, THEN
**IFERR**

If Error Conditional Structure Command  Starts IFERR ... THEN ... END and IFERR ... THEN ... ELSE ... END error trapping structures.

Keyboard Access:  `PRG` `NXT` `ERROR IFERR`

Affected by Flags:  Last Arguments (−55)

Remarks: Error trapping structures enable program execution to continue after a “trapped” error occurs.

- IFERR ... THEN ... END executes a sequence of commands if an error occurs. The syntax of IFERR ... THEN ... END is:

  ```plaintext
  IFERR trap-clause THEM error-clause END
  ```

  If an error occurs during execution of the trap clause:
  1. The error is ignored.
  2. The remainder of the trap clause is discarded.
  3. The key buffer is cleared.
  4. If any or all of the display is “frozen” (by FREEZE), that state is canceled.
  5. If Last Arguments is enabled, the arguments to the command that caused the error are returned to the stack.
  6. Program execution jumps to the error clause.

  The commands in the error clause are executed only if an error is generated during execution of the trap clause.

- IFERR ... THEN ... ELSE ... END executes one sequence of commands if an error occurs or another sequence of commands if an error does not occur. The syntax of IFERR ... THEN ... ELSE ... END is:

  ```plaintext
  IFERR trap-clause THEM error-clause ELSE normal-clause END
  ```

  If an error occurs during execution of the trap clause, the same six events listed above occur.

  If no error occurs, execution jumps to the normal clause at the completion of the trap clause.
Example: The following program uses IFFERR much like the built-in linear system of equations solver. The program takes a result vector and a matrix of coefficients and returns a least-squares solution to the equations.

```
  a b
  IFFERR a b / THEN LSQ END
```

Related Commands: CASE, ELSE, END, IF, THEN

---

**IFFT**

**Inverse Discrete Fourier Transform Command:** Computes the one- or two-dimensional inverse discrete Fourier transform of an array.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[array]₁</td>
<td>→</td>
<td>[array]₂</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH NXT FFT IFFT

**Affected by Flags:** None

**Remarks:** If the argument is an \( N \)-vector or an \( N \times 1 \) or \( 1 \times N \) matrix, IFFT computes the one-dimensional inverse transform. If the argument is an \( M \times N \) matrix, IFFT computes the two-dimensional inverse transform. \( M \) and \( N \) must be integral powers of 2.

The one-dimensional inverse discrete Fourier transform of an \( N \)-vector \( Y \) is the \( N \)-vector \( X \) where:

\[
X_n = \frac{1}{N} \sum_{k=0}^{N-1} Y_k e^{2\pi i k n / N}, \quad i = \sqrt{-1}
\]

for \( n = 0, 1, ..., N - 1 \).
**IFFT**

The two-dimensional inverse discrete Fourier transform of an $M \times N$ matrix $Y$ is the $M \times N$ matrix $X$ where:

$$X_{mn} = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} Y_{kl} e^{-\frac{2\pi ik m}{M}} e^{-\frac{2\pi jl n}{N}}, \quad i = \sqrt{-1}$$

for $m = 0, 1, ..., M-1$ and $n = 0, 1, ..., N-1$.

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The IFFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

**Related Commands:** FFT

---

**IFT**

**IF-THEN Command:** Executes obj if $T/F$ is nonzero. Discards obj if $T/F$ is zero.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
</table>
| $T/F$   | $\text{obj}$ | → | *It depends!*

**Keyboard Access:** [PRG] [BRCH] [NXT] [IFT]

**Affected by Flags:** None

**Remarks:** IFT lets you execute in stack syntax the decision-making process of the IF . . . THEN . . . END conditional structure. The “true clause” is obj in level 1.

**Example:** `× 0 "Positive" IFT` puts "Positive" in level 1 if $X$ contains a positive real number.

**Related Commands:** IFTE
IFTE

IF-THEN-ELSE Function: Executes the obj on level 2 if T/F is nonzero. Executes the obj on level 1 if T/F is zero.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/F</td>
<td>obj_{true}</td>
<td>obj_{false}</td>
<td>It depends!</td>
</tr>
</tbody>
</table>

Keyboard Access: PRG BRCH NXT IFTE

Affected by Flags: None

Remarks: IFTE lets you execute in stack syntax the decision-making process of the IF ... THEN ... ELSE ... END conditional structure. The "true clause" is obj_{true} in level 2. The "false clause" is obj_{false} in level 1.

IFTE is also allowed in algebraic expressions, with the following syntax:

\[ \text{IFTE}(\text{test}, \text{true-clause}, \text{false-clause}) \]

When an algebraic containing IFTE is evaluated, its first argument test is evaluated to a test result. If it returns a nonzero real number, true-clause is evaluated. If it returns zero, false-clause is evaluated.

Examples: The command sequence \( X \neq 0 \Rightarrow \text{"Positive"} \)
"Negative" IFTE leaves "Positive" on the stack if \( X \) contains a non-negative real number, or "Negative" if \( X \) contains a negative real number.

The algebraic \( \text{IFTE}(X \neq 0, \sin(x)/x, 1) \) returns the value of \( \sin(x)/x \), even for \( x = 0 \), which would normally cause an Infinite Result error.

Related Commands: IFT
**IM**

**Imaginary Part Function:** Returns the imaginary part of its (complex) argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>→</td>
<td>0</td>
</tr>
<tr>
<td>( (x, y) )</td>
<td>→</td>
<td>( y )</td>
</tr>
<tr>
<td>([ R-array ])</td>
<td>→</td>
<td>([ R-array ])</td>
</tr>
<tr>
<td>([ C-array ])</td>
<td>→</td>
<td>([ R-array ])</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'IM(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH NXT CMPL IM

**Affected by Flags:** Numerical Results \((-3)\)

**Remarks:** If the argument is an array, \( \text{IM} \) returns a real array, the elements of which are equal to the imaginary parts of the corresponding elements of the argument array. If the argument array is real, all of the elements of the result array are zero.

**Related Commands:** C→R, RE, R→C

**INCR**

**Increment Command:** Takes a variable on level 1, adds 1, stores the new value back into the original variable, and returns the new value to level 1.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td>( X_{\text{increment}} )</td>
</tr>
</tbody>
</table>
INDEP

Independent Variable Command: Specifies the independent variable and its plotting range.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'global'</td>
<td>→</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>{ global }</td>
<td>→</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>{ global \ x_{start} \ x_{end} }</td>
<td>→</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>{ \ x_{start} \ x_{end} }</td>
<td>→</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>\ x_{start}</td>
<td>→</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>\ x_{end}</td>
<td>→</td>
<td>←</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: ←MEMORY ARITH INCR

Affected by Flags: None

Remarks: The specification for the independent variable name and its plotting range is stored as the third parameter in the reserved variable PPAR. If the argument to INDEP is a:

- Global variable name, that name replaces the independent variable entry in PPAR.
- List containing a global name, that name replaces the independent variable name but leaves unchanged any existing plotting range.
- List containing a global name and two real numbers, that list replaces the independent variable entry.

Example: If 35.7 is stored in A, 'A' INCR returns 36.7.

Related Commands: DECR
INDEP

- List containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the independent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is X.

**Related Commands:** DEPND

INFORM

**User-Defined Dialog Box Command:** Creates a user-defined input form (dialog box).

<table>
<thead>
<tr>
<th>Lvl 5</th>
<th>Lvl 4</th>
<th>Lvl 3</th>
<th>Lvl 2</th>
<th>Lvl 1</th>
<th>Lvl 2</th>
<th>Lvl 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;title&quot; { s_1 \ s_2 \ ... \ s_n } format { resets } { init } \rightarrow { vals } 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;title&quot; { s_1 \ s_2 \ ... \ s_n } format { resets } { init } \rightarrow 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG NXT IN INFOR

**Affected by Flags:** None

**Remarks:** INFORM creates a standard dialog box based upon the following specifications:
<table>
<thead>
<tr>
<th>Variable</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;title&quot;</td>
<td>Title. This appears at the top of the dialog box.</td>
</tr>
<tr>
<td>{s_1 s_2 \ldots s_n}</td>
<td>Field definitions. A field definition (s_x) can have two formats: &quot;label&quot;, a field label, or { &quot;label&quot; &quot;helpInfo&quot; type_0 type_1 \ldots type_n }, a field label with optional help text that appears near the bottom of the screen, and an optional list of valid object types for that field. If object types aren't specified, all object types are valid. For information about object types, see the TYPE command. When creating a multi-column dialog box, you can span columns by using an empty list as a field definition. A field that appears to the left of an empty field automatically expands to fill the empty space.</td>
</tr>
<tr>
<td>format</td>
<td>Field format information. This is the number col or a list of the form { col tabs }: col is the number of columns the dialog box has, and tabs optionally specifies the number of tab stops between the labels and the highlighted fields. This list can be empty. col defaults to 1 and tabs defaults to 3.</td>
</tr>
<tr>
<td>{ resets }</td>
<td>Default values displayed when \textbf{RESET} is selected. Specify reset values displayed when the same order as the fields were specified. To specify no value, use the NOVAL command as a place holder. This list can be empty.</td>
</tr>
<tr>
<td>{ init }</td>
<td>Initial values displayed when the dialog box appears. Specify initial values displayed in the same order as the fields were specified. To specify no value, use the NOVAL command as a place holder. This list can be empty.</td>
</tr>
</tbody>
</table>

If you exit the dialog box by selecting \textbf{OK} or \textbf{ENTER}, INFORM returns the field values \{ vals \} on level 2, and puts a 1 on level 1. (If a field is empty, NOVAL is returned as a place holder.) If you exit the dialog box by selecting \textbf{CANCEL} or \textbf{CANCEL}, INFORM returns 0.
**Example:** If the following five lines are on the stack:

```
<table>
<thead>
<tr>
<th>Title text</th>
<th>Help text</th>
<th>Empty field</th>
<th>Field definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;The Title&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>{ &quot;ONE&quot; &quot;Name?&quot; 2 }</td>
<td>{ }</td>
<td>{ &quot;TWO&quot; &quot;Age?&quot; }</td>
</tr>
<tr>
<td></td>
<td>{ &quot;THREE&quot; &quot;Lucky numbers?&quot; 5 }</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Column count: {2}
Reset values: {NOVAL NOVAL {1 2 3}}
Allowed object type: {"Charlott" NOVAL {4 5 6}}

Pressing **INFORM** would produce:

![Image of the prompt window](image_url)

**Related Commands:** CHOOSE, INPUT, NOVAL, TYPE

**INPUT**

**Input Command:** Prompts for data input to the command line and prevents the user access to stack operations.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stack prompt&quot;</td>
<td>&quot;command-line prompt&quot;</td>
<td>→</td>
<td>&quot;result&quot;</td>
</tr>
<tr>
<td>&quot;stack prompt&quot;</td>
<td>{ \text{list}_{\text{command-line}} }</td>
<td>→</td>
<td>&quot;result&quot;</td>
</tr>
</tbody>
</table>
**Keyboard Access:**  [PRG] [NXT] [INPUT]

**Affected by Flags:**  None

**Remarks:** When INPUT is executed, the stack area is blanked and program execution is suspended for data input to the command line. The contents of "stack prompt" are displayed at the top of the stack area. Depending on the level 1 argument, the command line may also contain the contents of a string, or it may be empty. Pressing **ENTER** resumes program execution and returns the contents of the command line in string form to level 1.

In its general form, the level 1 argument for INPUT is a list that specifies the content and interpretation of the command line. The list can contain **one or more** of the following parameters, **in any order**:

- "command-line prompt", whose contents are placed in the command line for prompting when the program pauses.
- Either a **real number**, or a **list containing two real numbers**, that specifies the initial cursor position in the command line:
  - A real number \( n \) at the \( n \)th character from the left end of the first row (line) of the command line. A **positive** \( n \) specifies the insert cursor; a **negative** \( n \) specifies the replace cursor. \( \varnothing \) specifies the end of the command-line string.
  - A list that specifies the initial row and column position of the cursor: the first number in the list specifies a row in the command line (1 specifies the first row of the command line); the second number counts by characters from the left end of the specified line. \( \varnothing \) specifies the end of the command-line string in the specified row. A **positive** row number specifies the insert cursor; a **negative** row number specifies the replace cursor.
- One or more of the parameters **\texttt{ALG}**, \( \alpha \), or \( \forall \), entered as unquoted names:
  - **\texttt{ALG}** activates Algebraic/Program-entry mode.
  - \( \alpha \) \( (\alpha\rightarrow\mathbb{A}) \) specifies alpha lock.
  - \( \forall \) verifies if the characters in the result string "result", without the " delimiters, compose a valid object or objects. If the result-string characters do not compose a valid object or objects, INPUT displays the \texttt{Invalid Syntax} warning and prompts again for data.
**INPUT**

You can choose to specify as few as one of the level-1 list parameters. The default states for these parameters are:

- Blank command line.
- Insert cursor placed at the end of the command-line prompt string.
- Program-entry mode.
- Result string not checked for invalid syntax.

If you specify only a command-line prompt string for the level 1 argument, you don’t need to put it in a list.

**Related Commands:** PROMPT, STR→

---

**INV**

**Inverse (1/x) Analytic Function:** Returns the reciprocal or the matrix inverse.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>→</td>
<td>1/z</td>
</tr>
<tr>
<td>[[ matrix ]]</td>
<td>→</td>
<td>[[ matrix ]]^{-1}</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'INV(symb)'</td>
</tr>
<tr>
<td>x_unit</td>
<td>→</td>
<td>1/x_1/unit</td>
</tr>
</tbody>
</table>

**Keyboard Access:** 1/x

**Affected by Flags:** Numerical Results (−3)

**Remarks:** For a complex argument (x, y), the inverse is the complex number \( \left( \frac{x}{x^2+y^2}, \frac{-y}{x^2+y^2} \right) \).

Matrix arguments must be square (real or complex). The computed inverse matrix \( A^{-1} \) satisfies \( A \times A^{-1} = I_n \), where \( I_n \) is the \( n \times n \) identity matrix.

**Related Commands:** SINV, /
**IP**

**Integer Part Function:** Returns the integer part of its argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>→</td>
<td>$n$</td>
</tr>
<tr>
<td>$x_unit$</td>
<td>→</td>
<td>$n_unit$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'IP(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \[\text{MTH REAL NEXT IP}\]

**Affected by Flags:** Numerical Results (-3)

**Remarks:** The result has the same sign as the argument.

**Example:** 32.3 \_m IP returns 32 \_m.

**Related Commands:** FP

---

**IR**

**Infrared/Serial Transmission Command:** Directs I/O and printer output to the infrared or serial port ("wire").

**Keyboard Access:** \[\text{I/O IOPAR IR}\]

**Affected by Flags:** I/O Device (-33), Printing Device (-34)

**Remarks:** Toggles between IR and wire.

For more information, refer also to the reserved variable \textit{IOPAR (I/O parameters)} in appendix D, “Reserved Variables.”

**Related Commands:** BAUD, CKSM, PARITY, TRANSIO
**ISOL**

**Isolate Variable Command:** Returns an algebraic 'symb₂' that rearranges 'symb₁' to "isolate" the first occurrence of variable global.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb₁'</td>
<td>'global'→ 'symb₂'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** «(SYMBOLIC) ISOL

**Affected by Flags:** Principal Solution (−1), Numerical Results (−3)

When flag −3 is set, symbolic results are evaluated to real numbers. This means that the = sign is evaluated. If global or any other variable in the result equation is formal, an Undefined Name error results; if global and all other variables have values, a numerical result is returned from the calculation global − expression. This result has limited value. In general, execute ISOL with flag −3 clear.

**Remarks:** The result 'symb₂' is an equation of the form 'global=expression'. If global appears more than once, then 'symb₂' is effectively the right side of an equation obtained by rearranging and solving 'symb₁' to isolate the first occurrence of global on the left side of the equation.

If 'symb₁' is an expression, it is treated as the left side of an equation 'symb₁=0'.

If global appears in the argument of a function within 'symb₁', that function must be an analytic function—a function for which the HP 48 provides an inverse. Thus ISOL cannot solve 'IP(X)=0' for X, since IP has no inverse.

**Related Commands:** COLCT, EXPAN, QUAD, SHOW
**KERRM**

*Kermit Error Message Command:* Returns the text of the most recent Kermit error packet.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
</table>
|         | → | "error-message"

**Keyboard Access:** ← I/O NXT KERR

**Affected by Flags:** None

**Remarks:** If a Kermit transfer fails due to an error packet sent from the connected Kermit device to the HP 48, then executing KERRM retrieves and displays the error message. (Kermit errors not in packets are retrieved by ERRM rather than KERRM.)

**Related Commands:** FINISH, KGET, PKT, RECN, RECV, SEND, SERVER

---

**KEY**

*Key Command:* Returns to level 1 a test result and, if a key is pressed, returns to level 2 the row-column location $x_{nm}$ of that key.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>$x_{nm}$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>→</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG NXT IN KEY

**Affected by Flags:** None

**Remarks:** KEY returns a false result (0) to level 1 until a key is pressed. When a key is pressed, it returns a true result (1) to level...
KEY

1 and $x_{nm}$ to level 2. The result $x_{nm}$ is a two-digit number that identifies the row and column location of the key just pressed. Unlike WAIT, which returns a three-digit number that identifies alpha and shifted keyboard planes, KEY returns the row-column location of any key pressed, including $\leftarrow$, $\rightarrow$, and $\alpha$.

Example: The program $\texttt{« DO UNTIL KEY END 71 SAME »}$ returns 1 to the stack if the $\leftarrow$ key is pressed while the indefinite loop is running.

Related Commands: WAIT

KGET

Kermit Get Command: Used by a local Kermit to get a Kermit server to transmit the named object(s).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>&quot;name&quot;</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ name$<em>{old}$ name$</em>{new}$ }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ name$<em>{1}$ ... name$</em>{n}$ }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{{ name$<em>{old}$ name$</em>{new}$ } name ... }</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: $\leftarrow$I/O SRVR KGET

Affected by Flags: I/O Device (−33), RECV Overwrite (−36), I/O Messages (−39)

I/O Data Format (−35) also affects KGET as follows:

- If the server is an HP 48, then the server’s flag −35 affects KGET.
- If the server is not an HP 48 but the file being transferred originated from an HP 48, flag −35 has no effect.
- If the server is not an HP 48 and the file being transferred does not have the \%\%HP...; header, flag −35 tells the HP 48 whether to attempt parsing the incoming data.
Remarks: To rename an object when the local device gets it, include the old and new names in an embedded list. For example, `\{ \{ AAA BBB \} \} KGET` gets the variable named `AAA` but changes its name to `BBB`. `\{ \{ AAA BBB \} CCC \} KGET` gets `AAA` as `BBB` and gets `CCC` under its own name. (If the original name is not legal on the HP 48, enter it as a string.)

Related Commands: BAUD, CKSM, FINISH, PARITY, RECN, RECV, SEND, SERVER, TRANSIO

---

KILL

Cancel Halted Programs Command: Cancels all currently halted programs. (Halted programs are typically canceled by pressing `PRG NXT RUN KILL`.) If KILL is executed within a program, that program is also canceled.

Keyboard Access: `PRG NXT RUN KILL`

Affected by Flags: None

Remarks: Canceled programs cannot be resumed.

KILL cancels only halted programs and the program from which KILL was executed, if any. Commands that halt programs are HALT and PROMPT.

Suspended programs cannot be canceled. Commands that suspend programs are INPUT and WAIT.

Related Commands: CONT, DOERR, HALT, PROMPT
LABEL

**Label Axes Command:** Labels axes in *PICT* with *x*- and *y*-axis variable names and with the minimum and maximum values of the display ranges.

**Keyboard Access:** \( \leftarrow \text{PLOT} \text{NXT} \text{LABEL} \)

**Affected by Flags:** None

**Remarks:** The horizontal axis name is chosen in the following priority order:

1. If the *axes* parameter in the reserved variable *PPAR* is a list, then the *x*-axis element from that list is used.

2. If *axes* parameter is not a list, then the independent variable name in *PPAR* is used.

The vertical axis name is chosen in the following priority order:

1. If the *axes* parameter in *PPAR* is a list, then the *y*-axis element from that list is used.

2. If *axes* is not a list, then the dependent variable name from *PPAR* is used.

**Related Commands:** AXES, DRAW, DRAX

LAST

**Last Arguments Command:** Returns copies of the arguments of the most recently executed command.

**Keyboard Access:** None. Must be typed in.

**Remarks:** LAST is provided for compatibility with the HP 28S. LAST is the same as LASTARG. See LASTARG.
LASTARG

**Last Arguments Command:** Returns copies of the arguments of the most recently executed command.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level n</th>
<th>...</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>$obj_n$</td>
<td>...</td>
<td>$obj_1$</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

`→ ARG`  
`PRG NXT ERROR LASTA`

**Affected by Flags:** Last Arguments (−55)

**Remarks:** The objects return to the same stack levels that they originally occupied. Commands that take no arguments leave the current saved arguments unchanged.

When LASTARG follows a command that evaluates an algebraic expression or a program (as do $\partial$, $\int$, TAYLR, COLCT, DRAW, ROOT, ISOL, EVAL, and $\rightarrow$NUM), the last arguments saved are from the evaluated algebraic expression or program, not from the original command.

**Related Commands:** LAST

LCD

**LCD to Graphics Object Command:** Returns the current stack and menu display as a 131 x 64 graphics object.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>$grob$</td>
</tr>
</tbody>
</table>
**LCD**

**Keyboard Access:** PRG GRB NXT LCD

**Affected by Flags:** None

**Example:** LCD→ PICT STO PICTURE returns the current display to level 1 as a graphics object, stores it in PICT, then shows the image in the Picture environment.

**Related Commands:** →GROB, →LCD

---

**→LCD**

**Graphics Object to LCD Command:** Displays the graphics object from level 1, with its upper left pixel in the upper left corner of the display.

```
<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>grob</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>
```

**Keyboard Access:** PRG GRB NXT →LCD

**Affected by Flags:** None

**Remarks:** If the graphics object is larger than 131 x 56, it is truncated.

**Related Commands:** BLANK, →GROB, LCD→
LIBEVAL

Evaluate Library Function Command: Evaluates unnamed library functions.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n_function</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Using LIBEVAL with random addresses can corrupt memory. #n_function is of the form lllfffh, where lll is the library number, and fff the function number.

Related Commands: EVAL, SYSEVAL

LIBS

Libraries Command: Lists the title, number, and port of each library attached to the current directory.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>{ &quot;title&quot; n_lib n_port ... &quot;title&quot; n_lib n_port }</td>
</tr>
</tbody>
</table>

Keyboard Access: LIBRARY LIBS

Affected by Flags: None

Remarks: The title of a library often takes the form LIBRARY-NAME : Description. A library without a title is displayed as "".

Related Commands: ATTACH, DETACH
**LINE**

**Draw Line Command:** Draws a line in *PICT* between the coordinates in levels 1 and 2.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1 →</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x_1, y_1))</td>
<td>((x_2, y_2))</td>
<td>→</td>
</tr>
<tr>
<td>{ # n_1 # m_1 }</td>
<td>{ # n_2 # m_2 }</td>
<td>→</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(<\text{PRG} \text{ PICT \ LINE}>\)

**Affected by Flags:** None

**Example:** This program:

```
\( (0,0) \rightarrow (2,3) \text{ LINE } \{ \# \ 0d \# \ 0d \} \text{ PVIEW \ 7 FREEZE } \)
```
draws a line in *PICT* between two user-unit coordinates, displays *PICT* with pixel coordinate \( \{ \# \ 0d \# \ 0d \} \) at the upper left corner of the picture display, and freezes the display.

**Related Commands:** ARC, BOX, TLINE

---

**ΣLINE**

**Regression Model Formula Command:** Returns an expression representing the best fit line according to the current statistical model, using \( X \) as the independent variable name, and explicit values of the slope and intercept taken from the reserved variable \( \Sigma PAR \).

<table>
<thead>
<tr>
<th>Level 1 → Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow ) ( 'symb_{formula}' )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(<\text{STAT} \text{ FIT } \Sigma \text{ LINE}>\)
LINFIT

Affected by Flags: None

Remarks: For each curve fitting model, the following table indicates the form of the expression returned by $\Sigma$LINE, where $m$ is the slope, $x$ is the independent variable, and $b$ is the intercept.

<table>
<thead>
<tr>
<th>Model</th>
<th>Form of Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINFIT</td>
<td>$mx + b$</td>
</tr>
<tr>
<td>LOGFIT</td>
<td>$m \ln(x) + b$</td>
</tr>
<tr>
<td>EXPFIT</td>
<td>$b e^{mx}$</td>
</tr>
<tr>
<td>PWRFIT</td>
<td>$bx^m$</td>
</tr>
</tbody>
</table>

Example: If the current model is EXPFIT, and if the slope is 5 and the intercept 3, $\Sigma$LINE returns '3*EXP(5*X)'.

Related Commands: BESTFIT, COLΣ, CORR, COV, EXPFIT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, XCOL, YCOL

---

LINFIT

Linear Curve Fit Command: Stores LINFIT as the fifth parameter in the reserved variable $\Sigma PAR$, indicating that subsequent executions of LR are to use the linear curve fitting model.

Keyboard Access: $(\leftarrow)\text{STAT} \ \Sigma PAR \ \text{MODL} \ LINFIT$

Affected by Flags: None

Remarks: LINFIT is the default specification in $\Sigma PAR$. For a description of $\Sigma PAR$, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, EXPFIT, LOGFIT, LR, PWRFIT
**LININ**

**Linear Test Function:** Tests whether an algebraic is structurally linear for a given variable.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb'</td>
<td>'name'</td>
<td>→</td>
<td>0/1</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \[PRG\] [TEST] \([\text{PREV}\) [LININ]

**Affected by Flags:** None

**Remarks:** If any two subexpressions containing a variable (name) are combined only with addition and subtraction, and any subexpression containing the variable is at-most multiplied or divided by another factor not containing the variable, the algebraic ('symb') is determined to be linear for that variable.

LININ returns a 1 if the algebraic is linear for the variable, and a 0 if not.

**Example:**

'(X+1)*(Y-2^Z)+(X/(3-Z^8))'

'X'

LININ returns 1.

'(X^2-1)/(X+1)'  

'X'

LININ returns 0.

(Although this equation yields a linear equation when factored, LININ tests the equation as described above.)
**LIST→**

**List to Stack Command:** Takes a list of $n$ objects and returns them to separate levels, and returns the total number of objects to level 1.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$→$</th>
<th>Level n+1</th>
<th>...</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>${ obj_1 \ldots obj_n }$</td>
<td>$→$</td>
<td>$obj_1 \ldots$</td>
<td>$obj_n$</td>
<td>$n$</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** None. Must be typed in.

**Affected by Flags:** None

**Remarks:** The command OBJ→ also provides this function. LIST→ is included for compatibility with the HP 28S.

**Related Commands:** ARRY→, DTAG, EQ→, →LIST, OBJ→, STR→

---

**→LIST**

**Stack to List Command:** Takes $n$ objects from level n+1 through level 2 and returns a list of those $n$ objects.

<table>
<thead>
<tr>
<th>Level n+1</th>
<th>...</th>
<th>Level 2</th>
<th>Level 1</th>
<th>$→$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$obj_1 \ldots$</td>
<td>$obj_n$</td>
<td>$n$</td>
<td>$→$</td>
<td>${ obj_1 \ldots obj_n }$</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** `{PRG TYPE →LIST}

**Affected by Flags:** None

**Example:** The program

```
<< DEPTH →LIST 'A' STO >>
```

combines the entire contents of the stack into a list that is stored in variable $A$. 
\( \rightarrow \text{LIST} \)

**Related Commands:** \( \rightarrow \text{ARRY}, \rightarrow \text{LIST}, \rightarrow \text{STR}, \rightarrow \text{TAG}, \rightarrow \text{UNIT} \)

---

\( \sum \text{LIST} \)

**List Sum Command:** Returns the sum of the elements in a list.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ list }</td>
<td>( \rightarrow )</td>
<td>\text{sum}</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH} \ \sum \text{LIST} \)

**Affected by Flags:** None

**Remarks:** The elements in the list must be suitable for mutual addition.

**Examples:**
\( \{ 5, 8, 2 \} \ \sum \text{LIST} \) returns 15.
\( \{ A, B, C, 1 \} \ \sum \text{LIST} \) returns \('A+B+C+1'\).

**Related Commands:** \( \rightarrow \text{LIST}, \rightarrow \text{STREAM} \)

---

\( \Delta \text{LIST} \)

**List Differences Command:** Returns the first differences of the elements in a list.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ list }</td>
<td>( \rightarrow )</td>
<td>{ differences }</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH} \ \Delta \text{LIST} \)

**Affected by Flags:** None

3-170 Command Reference
**Remarks:** Adjacent elements in the list must be suitable for mutual subtraction.

**Examples:**
- `{ 4 2 0 1 7 6 0 9 1 } \(Δ\)LIST returns \(\{ 16 \ -19 \ 16 \ 43 \ 31 \}\).
- `{ A B C 1 2 3 } \(Δ\)LIST returns \(\{ 'B-A' 'C-B' '1-C' 1 1 \}\).
- `{ 'A+3' 'X/5' 'Y^4' } \(Δ\)LIST returns \(\{ 'X/5-(A+3)' 'Y^4-X/5' \}\).

**Related Commands:** \(Σ\)LIST, \(Δ\)LIST, STREAM

### List Product Command: \(π\)LIST

**Returns the product of the elements in a list.**

<table>
<thead>
<tr>
<th>Level 1</th>
<th>(→)</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>({ \text{list} })</td>
<td>(→)</td>
<td>\textit{product}</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] \(LIST \ πLIST\)

**Affected by Flags:** None

**Remarks:** The elements in the list must be suitable for mutual multiplication.

**Examples:**
- `{ 5 8 2 } \(π\)LIST returns 80.
- `{ A B C 1 } \(π\)LIST returns 'A*B*C'.

**Related Commands:** \(Σ\)LIST, \(Δ\)LIST, STREAM
**LN**

**Natural Logarithm Analytic Function:** Returns the natural (base $e$) logarithm of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>$\rightarrow$</td>
<td>$\ln z$</td>
</tr>
<tr>
<td>'symb'</td>
<td>$\rightarrow$</td>
<td>'LN(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** $\rightarrow$ LN

**Affected by Flags:** Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22)

**Remarks:** For $x=0$ or $(0, 0)$, an Infinite Result exception occurs, or, if flag −22 is set, −MAXR is returned.

The inverse of EXP is a *relation*, not a function, since EXP sends more than one argument to the same result. The inverse relation for EXP is expressed by ISOL as the general solution

$$'\ln(Z)+2\pi i n1'$$

The function LN is the inverse of a *part* of EXP, a part defined by restricting the domain of EXP such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of EXP are called the principal values of the inverse relation. LN in its entirety is called the principal branch of the inverse relation, and the points sent by LN to the boundary of the restricted domain of EXP form the branch cuts of LN.

The principal branch used by the HP 48 for LN was chosen because it is analytic in the regions where the arguments of the real-valued inverse function are defined. The branch cut for the complex-valued natural log function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of LN. The graph of the domain shows where the branch cut occurs: the heavy solid line marks...
one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation \( \text{LN}(\mathcal{Z}) + 2\pi i n_1 \) for the case \( n_1 = 0 \). For other values of \( n_1 \), the horizontal band in the lower graph is translated up (for \( n_1 \) positive) or down (for \( n_1 \) negative). Taken together, the bands cover the whole complex plane, which is the domain of \( \text{EXP} \).

You can view these graphs with domain and range reversed to see how the domain of \( \text{EXP} \) is restricted to make an inverse function possible. Consider the vertical band in the lower graph as the restricted domain \( \mathcal{Z} = (x, y) \). \( \text{EXP} \) sends this domain onto the whole complex plane in the range \( \mathcal{W} = (u, v) = \text{EXP}(x, y) \) in the upper graph.

**Related Commands:** ALOG, EXP, ISOL, LNP1, LOG
**LNP1**

**Natural Log of x Plus 1 Analytic Function:** Returns \( \ln(x + 1) \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( x )</th>
<th>( \ln(x+1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb'</td>
<td>( 'LNP1(symb)' )</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** 

**Affected by Flags:** Numerical Results (—3), Infinite Result Exception (—22)

**Remarks:** For values of \( x \) close to zero, \( 'LNP1(x)' \) returns a more accurate result than does \( '\ln(x+1)' \). Using LNP1 allows both the argument and the result to be near zero, and it avoids an intermediate result near 1. The calculator can express numbers within \( 10^{-449} \) of zero, but within only \( 10^{-11} \) of 1.

For values of \( x < -1 \), an Undefined Result error results. For \( x = -1 \), an Infinite Result exception occurs, or, if flag —22 is set, LNP1 returns \(-\text{MAXR}\).

**Related Commands:** EXPM, LN

---

**LOG**

**Common Logarithm Analytic Function:** Returns the common logarithm (base 10) of the argument.
LOG

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z )</td>
<td>→</td>
<td>( \log z )</td>
</tr>
<tr>
<td>( 'symb' )</td>
<td>→</td>
<td>( '\text{LOG}(symb)' )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \mathcal{R} \text{[LOG]} \)

**Affected by Flags:** Principal Solution \((-1)\), Numerical Results \((-3)\), Infinite Result Exception \((-22)\)

**Remarks:** For \( z = 0 \) or \((0, 0)\), an Infinite Result exception occurs, or, if flag \(-22\) is set (no error), LOG returns \(-\text{MAXR}\).

The inverse of ALOG is a relation, not a function, since ALOG sends more than one argument to the same result. The inverse relation for ALOG is expressed by ISOL as the general solution

\[ '\text{LOG}(z) + 2\pi i n \pi 1/2.30258509299' \]

The function LOG is the inverse of a part of ALOG, a part defined by restricting the domain of ALOG such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of ALOG are called the principal values of the inverse relation. LOG in its entirety is called the principal branch of the inverse relation, and the points sent by LOG to the boundary of the restricted domain of ALOG form the branch cuts of LOG.

The principal branch used by the HP 48 for LOG\((z)\) was chosen because it is analytic in the regions where the arguments of the real-valued function are defined. The branch cut for the complex-valued LOG function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

You can determine the graph for LOG\((z)\) from the graph for LN (see LN) and the relationship \( \log z = \ln z / \ln 10 \).

**Related Commands:** ALOG, EXP, ISOL, LN
**LOGFIT**

**Logarithmic Curve Fit Command:** Stores LOGFIT as the fifth parameter in the reserved variable $\Sigma PAR$, indicating that subsequent executions of LR are to use the logarithmic curve-fitting model.

**Keyboard Access:** $\leftarrow$STAT $\Sigma PAR$ MODL LOGFIT

**Affected by Flags:** None

**Remarks:** LINFIT is the default specification in $\Sigma PAR$. For a description of $\Sigma PAR$, see appendix D, “Reserved Variables.”

**Related Commands:** BESTFIT, EXPFIT, LINFIT, LR, PWRFIT

---

**LQ**

**LQ Factorization of a Matrix Command:** Returns the LQ factorization of an $n \times m$ matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[[ \text{matrix} ]]_A$</td>
<td>$[[ \text{matrix} ]]_L$</td>
<td>$[[ \text{matrix} ]]_Q$</td>
<td>$[[ \text{matrix} ]]_P$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH MATR FACTR LQ

**Affected by Flags:** None

**Remarks:** LQ factors an $m \times n$ matrix $A$ into three matrices:

- $L$ is a lower $m \times n$ trapezoidal matrix.
- $Q$ is an $n \times n$ orthogonal matrix.
- $P$ is a $m \times m$ permutation matrix.

Where $P \times A = L \times Q$.

**Related Commands:** LSQ, QR
**LR**

**Linear Regression Command:** Uses the currently selected statistical model to calculate the linear regression coefficients (intercept and slope) for the selected dependent and independent variables in the current statistics matrix (reserved variable $\Sigma DAT$).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 2</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Intercept: $x_1$</td>
<td></td>
<td>Slope: $x_2$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\leftrightarrow\) STAT FIT LR

**Affected by Flags:** None

**Remarks:** The columns of independent and dependent data are specified by the first two elements in the reserved variable $\Sigma PAR$, set by XCOL and YCOL, respectively. (The default independent and dependent columns are 1 and 2.) The selected statistical model is the fifth element in $\Sigma PAR$. LR stores the intercept and slope (untagged) as the third and fourth elements, respectively, in $\Sigma PAR$.

The coefficients of the exponential (EXPFIT), logarithmic (LOGFIT), and power (PWRFIT) models are calculated using transformations that allow the data to be fitted by standard linear regression. The equations for these transformations appear in the table below, where $b$ is the intercept and $m$ is the slope. The logarithmic model requires positive $x$-values (XCOL), the exponential model requires positive $y$-values (YCOL), and the power model requires positive $x$- and $y$-values.

For a description of $\Sigma PAR$, see appendix D, “Reserved Variables.”

**Transformation Equations**

<table>
<thead>
<tr>
<th>Model</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logarithmic</td>
<td>$y = b + m \ln(x)$</td>
</tr>
<tr>
<td>Exponential</td>
<td>$\ln(y) = \ln(b) + mx$</td>
</tr>
<tr>
<td>Power</td>
<td>$\ln(y) = \ln(b) + m \ln(x)$</td>
</tr>
</tbody>
</table>
LR

**Related Commands:** BESTFIT, COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, PREDX, PREDY, PWRFIT, XCOL, YCOL

---

**LSQ**

**Least Squares Solution Command:** Returns the minimum norm least squares solution to any system of linear equations where \( A \times X = B \).

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([ \text{array} ]_B)</td>
<td>([ \text{matrix} ]_A)</td>
<td>→</td>
<td>([ \text{array} ]_X)</td>
</tr>
<tr>
<td>([ \text{matrix} ]_B)</td>
<td>([ \text{matrix} ]_A)</td>
<td>→</td>
<td>([ \text{matrix} ]_X)</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- \( \text{SOLVE} \) \( \text{SYS} \) \( \text{LSQ} \)
- \( \text{MTH} \) \( \text{MATR} \) \( \text{LSQ} \)

**Affected by Flags:** Singular Values (-54)

**Remarks:** If \( B \) is a vector, the resulting vector has a minimum Euclidean norm \( \|X\| \) over all vector solutions that minimize the residual Euclidean norm \( \|A \times X - B\| \). If \( B \) is a matrix, each column of the resulting matrix, \( X_i \), has a minimum Euclidean norm \( \|X_i\| \) over all vector solutions that minimize the residual Euclidean norm \( \|A \times X_i - B_i\| \).

If \( A \) has less than full row rank (the system of equations is underdetermined), an infinite number of solutions exist. LSQ returns the solution with the minimum Euclidean length.

If \( A \) has less than full column rank (the system of equations is overdetermined), a solution that satisfies all the equations may not exist. LSQ returns the solution with the minimum residuals of \( A \times X - B \).

**Related Commands:** LQ, RANK, QR, /

3-178 Command Reference
MANT

LU Decomposition of a Square Matrix Command: Returns the LU decomposition of a square matrix.

Level 1 — Level 3 — Level 2 — Level 1

\[
\begin{align*}
[[ \text{matrix} ]_A & \rightarrow \quad [[ \text{matrix} ]_L \\
& \quad [[ \text{matrix} ]_U \quad \quad \quad \quad [[ \text{matrix} ]_P
\end{align*}
\]

Keyboard Access: \( \text{(MTH) MATR FACTR LU} \)

Affected by Flags: None

Remarks: When solving an exactly determined system of equations, inverting a square matrix, or computing the determinant of a matrix, the HP 48 factors a square matrix into its Crout LU decomposition using partial pivoting.

The Crout LU decomposition of \( A \) is a lower-triangular matrix \( L \), an upper-triangular matrix \( U \) with ones on its diagonal, and a permutation matrix \( P \), such that \( P \times A = L \times U \). The results satisfy \( P \times A \cong L \times U \).

Related Commands: DET, INV, LSQ, /

MANT

Mantissa Function: Returns the mantissa of the argument.

\[
\begin{align*}
\text{Level 1} & \rightarrow \quad \text{Level 1} \\
_\ast & \rightarrow \quad \text{y}_\text{mant} \\
'symb' & \rightarrow \quad '\text{MANT}(\text{symb})'
\end{align*}
\]

Keyboard Access: \( \text{(MTH) REAL NXT MANT} \)
MANT

Affected by Flags: Numerical Results (−3)

Example: \(-1.2\times10^3\) MANT returns 1.2.

Related Commands: SIGN, XPO

↑MATCH

Bottom-Up Match and Replace Command: Rewrites an expression.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→ Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb₁'</td>
<td>{'symb₀' 'symb₁'}</td>
<td>→ 'symb₂'</td>
<td>0/1</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>{'symb₀' 'symb₁' 'symb₂'}</td>
<td>→ 'symb₂'</td>
<td>0/1</td>
</tr>
</tbody>
</table>

Keyboard Access: (→)SYMBOLIC(NXT)↑MATCH

Affected by Flags: None

Remarks: ↑MATCH rewrites expressions or subexpressions that match a specified pattern 'symb₀'. An optional condition, 'symb₁', can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; 1 if it did, 0 if it did not.

The pattern 'symb₀' and replacement 'symb₁' can be normal expressions; for example, you can replace '\(\sin(\pi/6)\)' with '1/2'. You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with &; such as the name '&A', used in replacing '\(\sin(\&A+\pi)\)' with '−\(\sin(\&A)\)'. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↑MATCH works from bottom up; that is, it checks the lowest level (most deeply nested) subexpressions first. This approach works well for simplification. A subexpression simplified during one execution of ↑MATCH will be a simpler argument of its parent expression, so the parent expression can be simplified by another execution of ↑MATCH.
Several subexpressions can be simplified by one execution of \texttt{MATCH} provided none is a subexpression of any other.

**Examples:** This sequence:

\[
\text{\texttt{\texttt{'SIN(\pi/6)'} \{ \text{\texttt{'SIN(\pi/6) '1/2'} } \text{ \texttt{\uparrow MATCH}}
\]

returns \texttt{'1/2'} to level 2 and 1 (indicating a replacement was made) to level 1.

This sequence:

\[
\text{\texttt{\texttt{'SIN(X+\pi)'} \{ \text{\texttt{'SIN(\&A+\pi) ' -SIN(\&A)'} } \text{ \texttt{\uparrow MATCH}}
\]

returns \texttt{-SIN(X)} to level 2 and 1 to level 1.

This sequence:

\[
\text{\texttt{\texttt{'W+F(SQ(5))'} \{ \text{\texttt{'F(SQ\&A)'} ' \&A' ' \&A\neq 0'} } \text{ \texttt{\uparrow MATCH}}
\]

returns \texttt{'W+5'} to level 2 and 1 to level 1.

**Related Commands:** \texttt{\texttt{\texttt{\uparrow MATCH}}

\textbf{\texttt{\texttt{\uparrow MATCH}}

**Match Pattern Down Command:** Rewrites an expression.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{'symb_1'} { \texttt{'symb_pat' 'symb_repl'} }</td>
<td>\texttt{\rightarrow 'symb_2'} 0/1</td>
</tr>
<tr>
<td>\texttt{'symb_1'} { \texttt{'symb_pat' 'symb_repl' 'symb_cond'} }</td>
<td>\texttt{\rightarrow 'symb_2'} 0/1</td>
</tr>
</tbody>
</table>

\textbf{Keyboard Access:} \texttt{\uparrow SYMBOLIC} \texttt{\rightarrow SYMBOLIC} \texttt{\rightarrow MATCH}

\textbf{Affected by Flags:} None

\textbf{Remarks:} \texttt{\uparrow MATCH} rewrites expressions or subexpressions that match a specified pattern \texttt{'symb_pat'}. An optional condition, \texttt{'symb_cond'}, can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; 1 if it did, 0 if it did not.
The pattern \( \text{symb}_{\text{pat}} \) and replacement \( \text{symb}_{\text{repl}} \) can be normal expressions; for example, you can replace \( .5 \) with \( \sin(\frac{\pi}{6}) \). You can also use a “wildcard” in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with \&, such as the name \( \&A \), used in replacing \( \sin(\&A+\&B) \) with \( \sin(\&A)\cos(\&B)+\cos(\&A)\sin(\&B) \). Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

\MATCH\ works from top down; that is, it checks the entire expression first. This approach works well for expansion. An expression expanded during one execution of \MATCH\ will contain additional subexpressions, and those subexpressions can be expanded by another execution of \MATCH\. Several expressions can be expanded by one execution of \MATCH\ provided none is a subexpression of any other.

**Examples:**

\[ .5 \leftarrow .5 \ '\sin(\frac{\pi}{6})' \] \MATCH

returns \( \sin(\frac{\pi}{6}) \) to level 2 and 1 to level 1.

\[ \sin(U+V) \leftarrow \sin(\&A+\&B) \ 
\sin(\&A)\cos(\&B)+\cos(\&A)\sin(\&B) \] \MATCH

returns \( \sin(U)\cos(V)+\cos(U)\sin(V) \) to level 2 and 1 to level 1.

This sequence:

\[ \sin(5\times Z) \leftarrow \sin(\&A+\&B) \ 
\zeta(\&K=0,\&A,\text{comb}(\&A,\&K)\sin(K\times \pi)\ 
\cos(\&B^\times(\&A-\&K)\sin(\&B)^\times(\&K) \ 
\abs(\text{ip}(\&A))\equiv\&A \] \MATCH

returns

\[ \zeta(\&K=0,5,\text{comb}(5,\&K)\sin(K\times \pi)\cos(\&Z)\times(5-\&K)\sin(\&Z)^\times(\&K) \] to level 2 and 1 to level 1.

**Related Commands:** \MATCH
**MAX**

**Maximum Function:** Returns the greater (more positive) of the arguments.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y )</td>
<td>→</td>
<td>( \text{max}(x, y) )</td>
</tr>
<tr>
<td>( x )</td>
<td>'symb'</td>
<td>→</td>
<td>'MAX(x, symb)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>( x )</td>
<td>→</td>
<td>'MAX(symb, x)'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>'symb_2'</td>
<td>→</td>
<td>'MAX(symb_1, symb_2)'</td>
</tr>
<tr>
<td>( x_\text{unit}_1 )</td>
<td>( y_\text{unit}_2 )</td>
<td>→</td>
<td>( \text{max}(x_\text{unit}_1, y_\text{unit}_2) )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH REAL MAX

**Affected by Flags:** Numerical Results (-3)

**Examples:**
- \( 10 -23 \text{ MAX} \) returns 10.
- \( -10 -23 \text{ MAX} \) returns -10.
- \( 1\_\text{m} 9\_\text{cm} \text{ MAX} \) returns 1\_m.

**Related Commands:** MIN

---

**MAXR**

**Maximum Real Function:** Returns the symbolic constant 'MAXR' or its numerical representation, \( 9.99999999999\times 10^{499} \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>'MAXR'</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>9.99999999999E499</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH NXT CONS NXT MAXR
**MAXR**

**Affected by Flags:** Symbolic Constants (−2), Numerical Results (−3)

MAXR returns its numerical representation if flag −2 or −3 is set; otherwise, it returns its symbolic representation.

**Remarks:** MAXR is the largest numerical value that can be represented by the HP 48.

**Related Commands:** \( e, i, \text{MINR}, \pi \)

---

**MAXΣ**

**Maximum Sigma Command:** Finds the maximum coordinate value in each of the \( m \) columns of the current statistics matrix (reserved variable \( \Sigma DAT \)).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>( x_{\text{max}} )</td>
</tr>
<tr>
<td></td>
<td>→</td>
<td>( [x_{\text{max1}} \ x_{\text{max2}} \ \ldots \ x_{\text{maxm}}] )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( &\text{STAT} \ 1\text{VAR} \ \text{MAXΣ} \)

**Affected by Flags:** None

**Remarks:** The maxima are returned as a vector of \( m \) real numbers, or as a single real number if \( m = 1 \).

**Related Commands:** \( \text{BINS, MEAN, MINΣ, SDEV, TOT, VAR} \)
**MCALC**

**Make Calculated Value Command:** Designates a variable as a calculated value (not user-defined) for the Multiple-Equation Solver.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ list }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>&quot;ALL&quot;</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\leftarrow\)EQ\ LIB M\ E\ S M\ C\ A\ L\ C\ E\ R\

**Affected by Flags:** None

**Remarks:** MCALC designates a single variable, a list of variables, or all variables as calculated values.

**Related Commands:** MUSER

---

**MEAN**

**Mean Command:** Returns the mean of each of the \(m\) columns of coordinate values in the current statistics matrix (reserved variable \(\Sigma DAT\)).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>(x_{\text{mean}})</td>
</tr>
<tr>
<td></td>
<td>→</td>
<td>([x_{\text{mean}<em>1} \ x</em>{\text{mean}<em>2} \ldots \ x</em>{\text{mean}_m}])</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\leftarrow\)STAT \(1\)VAR MEAN

**Affected by Flags:** None
MEAN

Remarks: The mean is returned as a vector of \( m \) real numbers, or as a single real number if \( m = 1 \). The mean is computed from the formula:

\[
\frac{1}{n} \sum_{i=1}^{n} x_i
\]

where \( x_i \) is the \( i \)th coordinate value in a column, and \( n \) is the number of data points.

Related Commands: BINS, MAXΣ, MINΣ, SDEV, TOT, VAR

MEM

Memory Available Command: Returns the number of bytes of available RAM.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>x</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{MEM} \)

Affected by Flags: None

Remarks: The number returned is only a rough indicator of usable available memory, since recovery features (LASTARG, UNDO, and \( \text{CMD} \)) consume or release varying amounts of memory with each operation.

Before it can assess the amount of memory available, MEM must remove objects in temporary memory that are no longer being used. This clean-up process (also called “garbage collection”) also occurs automatically at other times when memory is full. Since this process can slow down calculator operation at undesired times, you can force it to occur at a desired time by executing MEM. In a program, execute MEM DROP.

Related Commands: BYTES
Display Menu Command: Displays a built-in menu or a library menu, or defines and displays a custom menu.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{\text{menu}}$</td>
<td>→</td>
</tr>
<tr>
<td>{ \textit{list definition} }</td>
<td>→</td>
</tr>
<tr>
<td>\texttt{&quot;name definition&quot;}</td>
<td>→</td>
</tr>
<tr>
<td>\texttt{obj}</td>
<td>→</td>
</tr>
</tbody>
</table>

Affected by Flags: None

Remarks: A built-in menu is specified by a real number $x_{\text{menu}}$. The format of $x_{\text{menu}}$ is $mm.pp$, where $mm$ is the menu number and $pp$ is the page of the menu. If $pp$ doesn’t correspond to a page of the specified menu, the first page is displayed. The following table lists the HP 48 built-in menus and the corresponding menu numbers.

<table>
<thead>
<tr>
<th>Menu #</th>
<th>Menu Name</th>
<th>Menu #</th>
<th>Menu Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Last Menu</td>
<td>15</td>
<td>MTH BASE</td>
</tr>
<tr>
<td>1</td>
<td>CST</td>
<td>16</td>
<td>MTH BASE LOGIC</td>
</tr>
<tr>
<td>2</td>
<td>VAR</td>
<td>17</td>
<td>MTH BASE BIT</td>
</tr>
<tr>
<td>3</td>
<td>MTH</td>
<td>18</td>
<td>MATH BASE BYTE</td>
</tr>
<tr>
<td>4</td>
<td>MTH VECTR</td>
<td>19</td>
<td>MTH FFT</td>
</tr>
<tr>
<td>5</td>
<td>MTH MATR</td>
<td>20</td>
<td>MTH CMPL</td>
</tr>
<tr>
<td>6</td>
<td>MTH MATR MAKE</td>
<td>21</td>
<td>MTH CONS</td>
</tr>
<tr>
<td>7</td>
<td>MTH MATR NORM</td>
<td>22</td>
<td>PRG</td>
</tr>
<tr>
<td>8</td>
<td>MTH MATR FACTR</td>
<td>23</td>
<td>PRG BRCH</td>
</tr>
<tr>
<td>9</td>
<td>MTH MATR COL</td>
<td>24</td>
<td>PRG BRCH IF</td>
</tr>
<tr>
<td>10</td>
<td>MTH MATR ROW</td>
<td>25</td>
<td>PRG BRCH CASE</td>
</tr>
<tr>
<td>11</td>
<td>MTH LIST</td>
<td>26</td>
<td>PRG BRCH START</td>
</tr>
<tr>
<td>12</td>
<td>MTH HYP</td>
<td>27</td>
<td>PRG BRCH FOR</td>
</tr>
<tr>
<td>13</td>
<td>MTH PROB</td>
<td>28</td>
<td>EDIT</td>
</tr>
<tr>
<td>14</td>
<td>MTH REAL</td>
<td>29</td>
<td>PRG BRCH DO</td>
</tr>
</tbody>
</table>

Command Reference 3-187
<table>
<thead>
<tr>
<th>Menu #</th>
<th>Menu Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>SOLVE ROOT SOLVR</td>
</tr>
<tr>
<td>31</td>
<td>PRG BRCH WHILE</td>
</tr>
<tr>
<td>32</td>
<td>PRG TEST</td>
</tr>
<tr>
<td>33</td>
<td>PRG TYPE</td>
</tr>
<tr>
<td>34</td>
<td>PRG LIST</td>
</tr>
<tr>
<td>35</td>
<td>PRG LIST ELEM</td>
</tr>
<tr>
<td>36</td>
<td>PRG LIST PROC</td>
</tr>
<tr>
<td>37</td>
<td>PRG GROB</td>
</tr>
<tr>
<td>38</td>
<td>PRG PICT</td>
</tr>
<tr>
<td>39</td>
<td>PRG IN</td>
</tr>
<tr>
<td>40</td>
<td>PRG OUT</td>
</tr>
<tr>
<td>41</td>
<td>PRG RUN</td>
</tr>
<tr>
<td>42</td>
<td>UNITS (Units Catalog Menu)</td>
</tr>
<tr>
<td>43</td>
<td>UNITS LENG</td>
</tr>
<tr>
<td>44</td>
<td>UNITS AREA</td>
</tr>
<tr>
<td>45</td>
<td>UNITS VOL</td>
</tr>
<tr>
<td>46</td>
<td>UNITS TIME</td>
</tr>
<tr>
<td>47</td>
<td>UNITS SPEED</td>
</tr>
<tr>
<td>48</td>
<td>UNITS MASS</td>
</tr>
<tr>
<td>49</td>
<td>UNITS FORCE</td>
</tr>
<tr>
<td>50</td>
<td>UNITS ENRG</td>
</tr>
<tr>
<td>51</td>
<td>UNITS POWR</td>
</tr>
<tr>
<td>52</td>
<td>UNITS PRESS</td>
</tr>
<tr>
<td>53</td>
<td>UNITS TEMP</td>
</tr>
<tr>
<td>54</td>
<td>UNITS ELEC</td>
</tr>
<tr>
<td>55</td>
<td>UNITS ANGL</td>
</tr>
<tr>
<td>56</td>
<td>UNITS LIGHT</td>
</tr>
<tr>
<td>57</td>
<td>UNITS RAD</td>
</tr>
<tr>
<td>58</td>
<td>UNITS VISC</td>
</tr>
<tr>
<td>59</td>
<td>UNITS</td>
</tr>
<tr>
<td>60</td>
<td>PRG ERROR IFERR</td>
</tr>
<tr>
<td>61</td>
<td>PRG ERROR</td>
</tr>
<tr>
<td>62</td>
<td>CHAR</td>
</tr>
<tr>
<td>63</td>
<td>MODES</td>
</tr>
<tr>
<td>64</td>
<td>MODES FMT</td>
</tr>
<tr>
<td>65</td>
<td>MODES ANGL</td>
</tr>
<tr>
<td>66</td>
<td>MODES FLAG</td>
</tr>
<tr>
<td>67</td>
<td>MODES KEYS</td>
</tr>
<tr>
<td>68</td>
<td>MODES MENU</td>
</tr>
<tr>
<td>69</td>
<td>MODES MISC</td>
</tr>
<tr>
<td>70</td>
<td>MEMORY</td>
</tr>
<tr>
<td>71</td>
<td>MEMORY DIR</td>
</tr>
<tr>
<td>72</td>
<td>MEMORY ARITH</td>
</tr>
<tr>
<td>73</td>
<td>STACK</td>
</tr>
<tr>
<td>74</td>
<td>SOLVE</td>
</tr>
<tr>
<td>75</td>
<td>SOLVE ROOT</td>
</tr>
<tr>
<td>76</td>
<td>SOLVE Diffe</td>
</tr>
<tr>
<td>77</td>
<td>SOLVE POLY</td>
</tr>
<tr>
<td>78</td>
<td>SOLVE SYS</td>
</tr>
<tr>
<td>79</td>
<td>SOLVE TVM</td>
</tr>
<tr>
<td>80</td>
<td>SOLVE TVM SOLVR</td>
</tr>
<tr>
<td>81</td>
<td>PLOT</td>
</tr>
<tr>
<td>82</td>
<td>PLOT PTYPE</td>
</tr>
<tr>
<td>83</td>
<td>PLOT PPAR</td>
</tr>
<tr>
<td>84</td>
<td>PLOT 3D</td>
</tr>
<tr>
<td>85</td>
<td>PLOT 3D PTYPE</td>
</tr>
<tr>
<td>86</td>
<td>PLOT 3D VPAR</td>
</tr>
<tr>
<td>87</td>
<td>PLOT STAT</td>
</tr>
<tr>
<td>88</td>
<td>PLOT STAT PTYPE</td>
</tr>
<tr>
<td>89</td>
<td>PLOT STAT ΣPAR</td>
</tr>
<tr>
<td>90</td>
<td>PLOT STAT ΣPAR MODL</td>
</tr>
<tr>
<td>91</td>
<td>PLOT STAT DATA</td>
</tr>
<tr>
<td>92</td>
<td>PLOT FLAG</td>
</tr>
<tr>
<td>93</td>
<td>SYMBOLIC</td>
</tr>
</tbody>
</table>

3-188 Command Reference
Library menus are specified in the same way as built-in menus, with the library number serving as the menu number.

Custom menus are specified by a list of the form `"label-object" action-object` (see appendix D, “Reserved Variables,” for details) or a name containing a list ("name:definition"). Either argument is stored in reserved variable `CST`, and the custom menu is subsequently displayed.

MENU takes any object as a valid argument and stores it in `CST`. However, the calculator can build a custom menu only if `CST` contains a list or a name containing a list. Thus, if an object other than a list or name containing a list is supplied to MENU, a Bad Argument Type error will occur when the calculator attempts to display the custom menu.

**Examples:**
- 5 MENU displays the first page of the MTH MATR NORM menu.
- 48.02 MENU displays the second page of the UNITS MASS menu.
- `{ A 123 "ABC" }` MENU displays the custom menu defined by the list argument.
- `'MYMENU'` MENU displays the custom menu defined by the name argument.
MENU

Related Commands: RCLMENU, TMENU

MERGE

Merge RAM Card Command: Merges the RAM from the card in port 1 with the rest of main user memory. Merged memory is no longer independent.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: MERGE is provided for compatibility with the HP 48S series. See MERGE1.

Related Commands: FREE1, MERGE1

MERGE1

Merge RAM Card Command: Merges the RAM from the card in port 1 with the rest of main user memory. Merged memory is no longer independent.

Keyboard Access: «LIBRARY» MERGE

Affected by Flags: None

Remarks: If the RAM card contains library or backup objects, they are moved to port 0 before the RAM is merged. Library and backup objects can exist only in independent memory (port 1 through 33 unmerged, or port 0).

Cards larger than 128K cannot be merged, and cannot be plugged into port 1.

3-190 Command Reference
Related Commands: FREE, FREE1, MERGE

**MIN**

**Minimum Function:** Returns the lesser (more negative) of its two arguments.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>→</td>
<td>min(x, y)</td>
</tr>
<tr>
<td>x</td>
<td>'symb'</td>
<td>→</td>
<td>'MIN(x, symb)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>x</td>
<td>→</td>
<td>'MIN(symb, x)'</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>'symb₂'</td>
<td>→</td>
<td>'MIN(symb₁, symb₂)'</td>
</tr>
</tbody>
</table>
| x
 characters  | y
 characters    | → | min(x
 characters, y
 characters)                  |

**Keyboard Access:** [MTH] REAL [MIN]

**Affected by Flags:** Numerical Results (-3)

**Examples:**
- 10 23 MIN returns 10.
- -10 -23 MIN returns -23.
- 1
   characters 9
   characters MIN returns 9
   characters.

**Related Commands:** MAX
MINEHUNT

MINEHUNT Game Command: Starts the MINEHUNT game.

Keyboard Access: `EQ LIB UTILS MINE`

Affected by Flags: None

Remarks: In the game, you are standing in the upper-left corner of an 8 x 16 battlefield grid. Your mission is to travel safely to the lower-right corner, avoiding invisible mines along the way. The game tells you how many mines are under the eight squares adjacent to your position.

Use the number or arrow keys to cross the battlefield one square at a time. (Use 7, 9, 1, and 3 to move diagonally.) You can exit the game any time by pressing [CANCEL].

To interrupt and save a game, press [STO]. This creates a variable `MHpar` in the current directory and ends the game. If `MHpar` exists when you next start Minehunt, the interrupted game resumes and `MHpar` is purged.

You can change the number of mines in the battlefield by creating a variable named `Nmines` containing the desired number. `Nmines` must contain a real number (1 to 64). If `Nmines` is negative, the mines are visible during the game.
**MINIT**

**Multiple Equation Menu Initialization Command:** Creates the reserved variable \( Mpar \).

**Keyboard Access:** \( \leftarrow \text{EQ LIB} \) \( \text{MES} \) \( \text{MINIT} \)

**Affected by Flags:** None

**Remarks:** MINIT takes multiple equations stored in \( EQ \) and creates the multiple equation reserved variable \( Mpar \). See appendix D, “Reserved Variables,” for information about \( Mpar \).

**Related Commands:** MITM, MROOT, MSOLVR

---

**MINR**

**Minimum Real Function:** Returns the symbolic constant 'MINR' or its numerical representation, \( 1.00000000000\times10^{-499} \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \rightarrow )</td>
<td>'MINR'</td>
</tr>
<tr>
<td></td>
<td>( \rightarrow )</td>
<td>( 1.00000000000\times10^{-499} )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH} \) \( \text{NXT} \) \( \text{CONS} \) \( \text{NXT} \) \( \text{MINR} \)

**Affected by Flags:** Symbolic Constants (−2), Numerical Results (−3)

MAXR returns its numerical representation if flag −2 or −3 is set; otherwise, it returns its symbolic representation.

**Remarks:** MINR is the smallest nonzero numerical value that can be represented by the HP 48.

**Related Commands:** \( e \), \( i \), MAXR, \( \pi \)
**MINΣ**

**Minimum Sigma Command:** Finds the minimum coordinate value in each of the $m$ columns of the current statistics matrix (reserved variable $Σ\text{DAT}$).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>$x_{\text{min}}$</td>
</tr>
<tr>
<td></td>
<td>→</td>
<td>$[x_{\text{min}1} \ x_{\text{min}2} \ ... \ x_{\text{min}m}]$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\leftrightarrow\)(STAT) 1VAR: MINΣ

**Affected by Flags:** None

**Remarks:** The minima are returned as a vector of $m$ real numbers, or as a single real number if $m = 1$.

**Related Commands:** BINS, MAXΣ, MEAN, SDEV, TOT, VAR

---

**MITM**

**Multiple Equation Menu Item Order Command:** Changes multiple equation menu titles and order.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;title&quot;</td>
<td>{list}</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\leftrightarrow\)(EQ LIB) 1VAR: MITM

**Affected by Flags:** None

**Remarks:** list contains the variable names in the order you want. Use "" to indicate a blank label. You must include all variables in the original menu and no others.
**MROOT**

**Related Commands:** MINIT

---

**MOD**

**Modulo Function:** Returns a remainder defined by:

\[ x \mod y = x - y \lfloor \frac{x}{y} \rfloor \]

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1 →</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y ) →</td>
<td>( x \mod y )</td>
</tr>
<tr>
<td>( x )</td>
<td>'symb' →</td>
<td>'MOD(x, symb)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>( x ) →</td>
<td>'MOD(symb, x)'</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>'symb₂' →</td>
<td>'MOD(symb₁, symb₂)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [REAL] [MOD]

**Affected by Flags:** Numerical Results (-3)

**Remarks:** Mod \((x, y)\) is periodic in \(x\) with period \(y\). Mod \((x, y)\) lies in the interval \([0, y)\) for \(y > 0\) and in \((y, 0]\) for \(y < 0\).

**Related Commands:** FLOOR, /

---

**MROOT**

**Multiple Roots Command:** Uses the Multiple-Equation Solver to solve for one or more variables using the equation set in \(Mpar\).

<table>
<thead>
<tr>
<th>Level 1 →</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name' →</td>
<td>( x )</td>
</tr>
<tr>
<td>&quot;ALL&quot; →</td>
<td></td>
</tr>
</tbody>
</table>
MROOT

Keyboard Access: (→)(EQLIB) MEX MROO

Affected by Flags: None

Remarks: Solves for one or more variables starting with only user-defined values, and leaves found values in the variables. No status messages are displayed. Given a variable name, MROOT returns the found value; it can also take "ALL" (stores a found value for each variable) and return nothing to the stack.

Related Commands: MCALC, MUSER

MSGBOX

Message Box Command: Creates a user-defined message box.

{ }  

Level 1 → Level 1

"message" →

Keyboard Access: (PRG)(NXT) OUT MSGB

Affected by Flags: None

Remarks: MSGBOX displays "message" in the form of a standard message box. Message text longer than 75 characters (including spaces) is truncated to 75 characters. You can use spaces and new-line characters ((→)(e=#)) to control word-wrapping and line breaks within the message.

Program execution resumes when the message box is exited by selecting OK or CANCEL.

Related Commands: CHOOSE, INFORM, PROMPT
**MSOLVR**

**Multiple-Equation Solver Command:** Gets the Multiple-Equation Solver variable menu for the set of equations defined by \( Mpar \).

**Keyboard Access:**

\[ \text{EQ LIB} \quad \text{MES} \quad \text{MSOL} \]
\[ \text{EQ LIB} \quad \text{EQ LIB} \quad \text{MSOL} \]

**Affected by Flags:** None

**Remarks:** The Multiple-Equation Solver application can solve a set of two or more equations for unknown variables by finding the roots of each equation, one at a time.

The Multiple-Equation Solver uses the list of equations stored in \( EQ \). “Equations” in this context includes programs, expressions, and variable names that evaluate to a single value. The Multiple-Equation Solver requires that \( EQ \) contain more than one equation—that is, the HP Solve application would include the \( N \times \text{EQ} \) menu label for \( EQ \). The solver uses \( EQ \) to create a reserved variable \( Mpar \) that is used during the solution process. \( Mpar \) contains the equation set plus additional information. See appendix D, “Reserved Variables,” for information about \( Mpar \).

**Related Commands:** EQNLIB, SOLVEQN

---

**MUSER**

**Make User-Defined Variable Command:** Designates a variable as user-defined for the Multiple-Equation Solver.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{‘name’}</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ \text{list} }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>&quot;ALL&quot;</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>
**MUSER**

**Keyboard Access:** `EQLIB MUSER`

**Affected by Flags:** None

**Remarks:** MUSER designates a single variable, a list of variables, or all variables as user-defined.

**Related Commands:** MCALC

---

**NDIST**

**Normal Distribution Command:** Returns the normal probability distribution (bell curve) at $x$ based on the mean $m$ and variance $v$ of the normal distribution.

![Level 3 Level 2 Level 1 → Level 1](image)

$$m \quad v \quad x \quad \rightarrow \quad \text{ndist}(m, v, x)$$

**Keyboard Access:** `MTH NXT PROB NXT NDIST`

**Affected by Flags:** None

**Remarks:** NDIST is calculated using this formula:

$$
\text{ndist}(m, v, x) = e^{-\frac{(x-m)^2}{2v}} \frac{1}{\sqrt{2\pi v}}
$$

**Related Commands:** UTPN
Negate Analytic Function: Changes the sign or negates an object.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>$→$</td>
<td>$−z$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>$→$</td>
<td>$−n_2$</td>
</tr>
<tr>
<td>[ array ]</td>
<td>$→$</td>
<td>[ $−$ array ]</td>
</tr>
<tr>
<td>'symb'</td>
<td>$→$</td>
<td>'−(symb)'</td>
</tr>
<tr>
<td>$x_unit$</td>
<td>$→$</td>
<td>$−x_unit$</td>
</tr>
<tr>
<td>grobl</td>
<td>$→$</td>
<td>grob2</td>
</tr>
<tr>
<td>$PICT_1$</td>
<td>$→$</td>
<td>$PICT_2$</td>
</tr>
</tbody>
</table>

Keyboard Access:

$+/-$

MTH NXT CMPL NXT NEG

Affected by Flags: Numerical Results ($-3$), Binary Integer Wordsize ($-5$ through $-10$)

Remarks: Negating an array creates a new array containing the negative of each of the original elements. Negating a binary number takes its two’s complement (complements each bit and adds 1).

Negating a graphics object “inverts” it (toggles each pixel from on to off, or vice-versa). If the argument is $PICT$, the graphics object stored in $PICT$ is inverted.

Related Commands: ABS, CONJ, NOT, SIGN
NEWOB

New Object Command: Creates a new copy of the specified object.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>→</td>
<td>obj</td>
</tr>
</tbody>
</table>

Keyboard Access: («)(MEMORY) NEWOB

Affected by Flags: Last Arguments (−55)

In order for NEWOB to immediately release the memory occupied by the original copy, flag −55 must be set so that the copy is not saved as a last argument.

Remarks: NEWOB has two main uses:

- NEWOB enables the purging of a library or backup object that has been recalled from a port. NEWOB creates a new, separate copy of the object in memory, thereby allowing the original copy to be purged.
- Creating a new copy of an object that originated in a larger composite object (such as a list) allows you to recover the memory associated with the larger object when that larger object is no longer needed.

Examples: 

:0: BKUP1 RCL NEWOB :0: BKUP1 PURGE recalls and purges the backup object BKUP1.

3 GET NEWOB retrieves the third element out of a list in the stack, recovering the memory occupied by the whole list.

Related Commands: MEM, PURGE
**NEXT**

**NEXT Command:** Ends definite loop structures.

See the FOR and START command entries for syntax information.

**Keyboard Access:**

- PRG BRCH START BRCH
- PRG BRCH FOR NEXT

**Remarks:** See the FOR and START keyword entries for more information.

**Related Commands:** FOR, START, STEP

---

**NEXT**

**Next Operation:** Returns but does not execute the next one or two steps of a program.

**Keyboard Access:**

- PRG NXT RUN NEXT

**Affected by Flags:** None

**Related Commands:** SST, SST|

---

**NOT**

**NOT Command:** Returns the one's complement or logical inverse of the argument.
NOT

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
<tr>
<td>T/F</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>&quot;string₁&quot;</td>
<td>→</td>
<td>&quot;string₂&quot;</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'NOT symb'</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

PRG TEST NXT NOT

MTH BASE NXT LOGIC NOT

**Affected by Flags:** Numerical Results (–3), Binary Integer Wordsize (–5 through –10)

**Remarks:** When the argument is a binary integer or string, NOT complements each bit in the argument to produce the result.

- A binary integer is treated as a sequence of bits as long as the current wordsize.
- A string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code).

When the argument is a real number or symbolic, NOT does a true/false test. The result is 1 (true) if the argument is zero; it is 0 (false) if the argument is nonzero. This test is usually done on a test result (T/F).

If the argument is an algebraic object, then the result is an algebraic of the form 'NOT symb'. Execute →NUM (or set flag –3 before executing NOT) to produce a numeric result from the algebraic result.

**Related Commands:** AND, OR, XOR
NOVAL

INFORM Place Holder/Result Command: Place holder for reset and initial values in user-defined dialog boxes. NOVAL is returned to the stack when a field is empty.

Keyboard Access: PRG NXT IN NOVA

Affected by Flags: None

Remarks: NOVAL is used to mark an empty field in a user-defined dialog box created with the INFORM command. INFORM defines fields sequentially. If default values are used for those fields, the defaults must be defined in the same order as the fields were defined. To skip over (not provide defaults for) some of the fields, use the NOVAL command.

After INFORM terminates, NOVAL is returned to the stack (on level 2) if a field is empty and OK or ENTER is selected.

Related Commands: INFORM

NSUB

Number of Sublist Command: Provides a way to access the current sublist position during an iteration of a program or command applied using DOSUBS.

Keyboard Access: PRG LIST PROC NSUB

Affected by Flags: None

Remarks: Returns an Undefined Local Name error if executed when DOSUBS is not active.

Related Commands: DOSUBS, ENDSUB
NUM

Character Number Command: Returns the character code \( n \) for the first character in the string.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;string&quot;</td>
<td>( \rightarrow )</td>
<td>( n )</td>
</tr>
</tbody>
</table>

Keyboard Access:

\[ \text{CHARS} \text{ NUM} \]
\[ \text{PRG TYPE NXT NUM} \]

Affected by Flags: None

Remarks: The character codes are an extension of ISO 8859/1. Codes 128 through 159 are unique to the HP 48.

The following tables show the relation between character codes (results of NUM, arguments to CHR) and characters (results of CHR, arguments to NUM).
<table>
<thead>
<tr>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>32</td>
<td></td>
<td>64</td>
<td>@</td>
<td>96</td>
<td>'</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>33</td>
<td>!</td>
<td>65</td>
<td>A</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>34</td>
<td>&quot;</td>
<td>66</td>
<td>B</td>
<td>98</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>35</td>
<td>#</td>
<td>67</td>
<td>C</td>
<td>99</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>36</td>
<td>$</td>
<td>68</td>
<td>D</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>37</td>
<td>%</td>
<td>69</td>
<td>E</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>38</td>
<td>&amp;</td>
<td>70</td>
<td>F</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>39</td>
<td>'</td>
<td>71</td>
<td>G</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>40</td>
<td>(</td>
<td>72</td>
<td>H</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>41</td>
<td>)</td>
<td>73</td>
<td>I</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>42</td>
<td>*</td>
<td>74</td>
<td>J</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>43</td>
<td>+</td>
<td>75</td>
<td>K</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>44</td>
<td>,</td>
<td>76</td>
<td>L</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>45</td>
<td>-</td>
<td>77</td>
<td>M</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>46</td>
<td>.</td>
<td>78</td>
<td>N</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>47</td>
<td>/</td>
<td>79</td>
<td>O</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>48</td>
<td>0</td>
<td>80</td>
<td>P</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>49</td>
<td>1</td>
<td>81</td>
<td>Q</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>50</td>
<td>2</td>
<td>82</td>
<td>R</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>51</td>
<td>3</td>
<td>83</td>
<td>S</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>52</td>
<td>4</td>
<td>84</td>
<td>T</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>53</td>
<td>5</td>
<td>85</td>
<td>U</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>54</td>
<td>6</td>
<td>86</td>
<td>V</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>55</td>
<td>7</td>
<td>87</td>
<td>W</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>56</td>
<td>8</td>
<td>88</td>
<td>X</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>57</td>
<td>9</td>
<td>89</td>
<td>Y</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>58</td>
<td>;</td>
<td>90</td>
<td>Z</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>59</td>
<td>&lt;</td>
<td>91</td>
<td>[</td>
<td>123</td>
<td>c</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>60</td>
<td>&gt;</td>
<td>92</td>
<td>\</td>
<td>124</td>
<td>l</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>61</td>
<td>=</td>
<td>93</td>
<td>]</td>
<td>125</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>62</td>
<td>?</td>
<td>94</td>
<td>^</td>
<td>126</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>63</td>
<td>\</td>
<td>95</td>
<td>_</td>
<td>127</td>
<td></td>
</tr>
</tbody>
</table>
### Character Codes (128 — 255)

<table>
<thead>
<tr>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>å</td>
<td>160</td>
<td>i</td>
<td>192</td>
<td>À</td>
<td>224</td>
<td>å</td>
</tr>
<tr>
<td>129</td>
<td>å</td>
<td>161</td>
<td>j</td>
<td>193</td>
<td>À</td>
<td>225</td>
<td>å</td>
</tr>
<tr>
<td>130</td>
<td>Å</td>
<td>162</td>
<td>ő</td>
<td>194</td>
<td>À</td>
<td>226</td>
<td>å</td>
</tr>
<tr>
<td>131</td>
<td>Å</td>
<td>163</td>
<td>ü</td>
<td>195</td>
<td>À</td>
<td>227</td>
<td>å</td>
</tr>
<tr>
<td>132</td>
<td>À</td>
<td>164</td>
<td>é</td>
<td>196</td>
<td>À</td>
<td>228</td>
<td>å</td>
</tr>
<tr>
<td>133</td>
<td>à</td>
<td>165</td>
<td>€</td>
<td>197</td>
<td>À</td>
<td>229</td>
<td>å</td>
</tr>
<tr>
<td>134</td>
<td>à</td>
<td>166</td>
<td>ł</td>
<td>198</td>
<td>Æ</td>
<td>230</td>
<td>æ</td>
</tr>
<tr>
<td>135</td>
<td>À</td>
<td>167</td>
<td>$</td>
<td>199</td>
<td>Ç</td>
<td>231</td>
<td>ç</td>
</tr>
<tr>
<td>136</td>
<td>À</td>
<td>168</td>
<td>..</td>
<td>200</td>
<td>Æ</td>
<td>232</td>
<td>æ</td>
</tr>
<tr>
<td>137</td>
<td>Å</td>
<td>169</td>
<td>ø</td>
<td>201</td>
<td>Æ</td>
<td>233</td>
<td>æ</td>
</tr>
<tr>
<td>138</td>
<td>À</td>
<td>170</td>
<td>å</td>
<td>202</td>
<td>Æ</td>
<td>234</td>
<td>æ</td>
</tr>
<tr>
<td>139</td>
<td>À</td>
<td>171</td>
<td>«</td>
<td>203</td>
<td>Æ</td>
<td>235</td>
<td>æ</td>
</tr>
<tr>
<td>140</td>
<td>À</td>
<td>172</td>
<td>-</td>
<td>204</td>
<td>Æ</td>
<td>236</td>
<td>æ</td>
</tr>
<tr>
<td>141</td>
<td>À</td>
<td>173</td>
<td>-</td>
<td>205</td>
<td>Æ</td>
<td>237</td>
<td>æ</td>
</tr>
<tr>
<td>142</td>
<td>À</td>
<td>174</td>
<td>Ø</td>
<td>206</td>
<td>Æ</td>
<td>238</td>
<td>æ</td>
</tr>
<tr>
<td>143</td>
<td>À</td>
<td>175</td>
<td>-</td>
<td>207</td>
<td>Æ</td>
<td>239</td>
<td>æ</td>
</tr>
<tr>
<td>144</td>
<td>À</td>
<td>176</td>
<td>-</td>
<td>208</td>
<td>Æ</td>
<td>240</td>
<td>æ</td>
</tr>
<tr>
<td>145</td>
<td>À</td>
<td>177</td>
<td>i</td>
<td>209</td>
<td>Æ</td>
<td>241</td>
<td>æ</td>
</tr>
<tr>
<td>146</td>
<td>À</td>
<td>178</td>
<td>$</td>
<td>210</td>
<td>Æ</td>
<td>242</td>
<td>æ</td>
</tr>
<tr>
<td>147</td>
<td>À</td>
<td>179</td>
<td>ë</td>
<td>211</td>
<td>Æ</td>
<td>243</td>
<td>æ</td>
</tr>
<tr>
<td>148</td>
<td>À</td>
<td>180</td>
<td>-</td>
<td>212</td>
<td>Æ</td>
<td>244</td>
<td>æ</td>
</tr>
<tr>
<td>149</td>
<td>À</td>
<td>181</td>
<td>µ</td>
<td>213</td>
<td>Æ</td>
<td>245</td>
<td>æ</td>
</tr>
<tr>
<td>150</td>
<td>À</td>
<td>182</td>
<td>¶</td>
<td>214</td>
<td>Æ</td>
<td>246</td>
<td>æ</td>
</tr>
<tr>
<td>151</td>
<td>À</td>
<td>183</td>
<td>-</td>
<td>215</td>
<td>Æ</td>
<td>247</td>
<td>æ</td>
</tr>
<tr>
<td>152</td>
<td>À</td>
<td>184</td>
<td>-</td>
<td>216</td>
<td>Æ</td>
<td>248</td>
<td>æ</td>
</tr>
<tr>
<td>153</td>
<td>À</td>
<td>185</td>
<td>.</td>
<td>217</td>
<td>Æ</td>
<td>249</td>
<td>æ</td>
</tr>
<tr>
<td>154</td>
<td>À</td>
<td>186</td>
<td>-</td>
<td>218</td>
<td>Æ</td>
<td>250</td>
<td>æ</td>
</tr>
<tr>
<td>155</td>
<td>À</td>
<td>187</td>
<td>»</td>
<td>219</td>
<td>Æ</td>
<td>251</td>
<td>æ</td>
</tr>
<tr>
<td>156</td>
<td>À</td>
<td>188</td>
<td>-</td>
<td>220</td>
<td>Æ</td>
<td>252</td>
<td>æ</td>
</tr>
<tr>
<td>157</td>
<td>À</td>
<td>189</td>
<td>-</td>
<td>221</td>
<td>Æ</td>
<td>253</td>
<td>æ</td>
</tr>
<tr>
<td>158</td>
<td>À</td>
<td>190</td>
<td>-</td>
<td>222</td>
<td>Æ</td>
<td>254</td>
<td>æ</td>
</tr>
<tr>
<td>159</td>
<td>À</td>
<td>191</td>
<td>-</td>
<td>223</td>
<td>Æ</td>
<td>255</td>
<td>æ</td>
</tr>
</tbody>
</table>

**Related Commands:**  CHR, POS, REPL, SIZE, SUB

**3-206 Command Reference**
NUMX

**Evaluate to Number Command:** Evaluates a symbolic argument object (other than a list) and returns the numerical result.

\[
\begin{array}{|c|c|}
\hline
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
obj_{sym} & \rightarrow & z \\
\hline
\end{array}
\]

**Keyboard Access:** \( \leftarrow \rightarrow \text{NUM} \)

**Affected by Flags:** None

**Remarks:** \( \rightarrow \text{NUM} \) repeatedly evaluates a symbolic argument until a numerical result is achieved. The effect is the same as evaluating the symbolic argument in Numerical Result mode (flag \(-3\) set).

**Related Commands:** EVAL, SYSEVAL

NUMX

**Number of X-Steps Command:** Sets the number of x-steps for each y-step in 3D perspective plots.

\[
\begin{array}{|c|c|}
\hline
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
n_x & \rightarrow & \\
\hline
\end{array}
\]

**Keyboard Access:** \( \leftarrow \text{PLOT} \ \text{NXT} \ \text{3D} \ \text{VPAR} \ \text{NXT} \ \text{NUMX} \)

**Affected by Flags:** None

**Remarks:** The number of x-steps is the number of independent variable points plotted for each dependent variable point plotted. This number must be 2 or more. This value is stored in the reserved
variable VPAR. YSLICE is the only 3D plot type that does not use this value.

**Related Commands:** NUMY

---

**NUMY**

**Number of Y-Steps Command:** Sets the number of y-steps across the view volume in 3D perspective plots.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_y )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{←PLOT NXT 3D VPAR NXT NUMY} \)

**Affected by Flags:** None

**Remarks:** The number of y-steps is the number of dependent variable points plotted across the view volume. This number must be 2 or more. This value is stored in the reserved variable VPAR.

**Related Commands:** NUMX

---

**NZ**

**Number of Rows Command:** Returns the number of rows in the current statistical matrix (reserved variable \( \Sigma DAT \)).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>( n_{rows} )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{←STAT SUMS NZ} \)
Object to Stack Command: Separates an object into its components onto the stack. For some object types, the number of components is returned to level 1.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level n+1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x,y)</td>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>{obj_1</td>
<td></td>
<td>obj_1</td>
<td>n</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>obj_n</td>
<td></td>
<td>obj_n</td>
<td></td>
</tr>
<tr>
<td>[x_1</td>
<td></td>
<td>x_1</td>
<td>x_n</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>x_n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[[x_{11}</td>
<td></td>
<td>x_{11}</td>
<td>x_{m,n}</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>x_{m,n}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;obj&quot;</td>
<td></td>
<td></td>
<td>evaluated-object</td>
</tr>
<tr>
<td>'symb'</td>
<td></td>
<td>arg_1</td>
<td>arg_n</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'function'</td>
</tr>
<tr>
<td>x_unit</td>
<td></td>
<td>x</td>
<td>1_unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:tag:obj</td>
<td></td>
<td>obj</td>
<td>&quot;tag&quot;</td>
</tr>
</tbody>
</table>

Keyboard Access:

- CHAR3 NXT OBJ+
- PRG TYPE OBJ+

Affected by Flags: None

Remarks: If the argument is a complex number, list, array, or string, OBJ→ provides the same functions as C→R, LIST→, ARRY→, and STR→, respectively. For lists, OBJ→ also returns the number of list elements. If the argument is an array, OBJ→ also returns the dimensions \{m n\} of the array, where \(m\) is the number of rows and \(n\) is the number of columns.

For algebraic objects, OBJ→ returns the arguments of the top-level (least-nested) function \(\text{arg}_1 \ldots \text{arg}_n\), the number of arguments.
OBJ→

of the top-level function \( n \), and the name of the top-level function \( \text{function} \).

If the argument is a string, the object sequence defined by the string is executed.

**Example:** The command sequence `\( \int (0, 1, \sin(x), x) \)` OBJ→ returns:

<table>
<thead>
<tr>
<th>6:</th>
<th>0</th>
<th>first argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>5:</td>
<td>1</td>
<td>second argument</td>
</tr>
<tr>
<td>4:</td>
<td>'( \sin(x) )'</td>
<td>third argument</td>
</tr>
<tr>
<td>3:</td>
<td>'x'</td>
<td>fourth argument</td>
</tr>
<tr>
<td>2:</td>
<td>4</td>
<td>number of arguments for ( \int )</td>
</tr>
<tr>
<td>1:</td>
<td>( \int )</td>
<td>function name</td>
</tr>
</tbody>
</table>

**Related Commands:** ARRY→, C→R, DTAG, EQ→, LIST→, R→C, STR→, →TAG

---

**OCT**

**Octal Mode Command:** Selects octal base for binary integer operations. (The default base is decimal.)

**Keyboard Access:** [MTH] BASE OCT

**Affected by Flags:** Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

**Remarks:** Binary integers require the prefix #. Binary integers entered and returned in octal base automatically show the suffix \( o \). If the current base is not octal, enter an octal number by ending it with \( o \). It will be displayed in the current base when entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

**Related Commands:** BIN, DEC, HEX, RCWS, STWS
**OFF**

**Off Command:** Turns off the calculator.

**Keyboard Access:** \( \text{PRG} \) \( \text{NXT} \) \( \text{RUN} \) \( \text{NXT} \) \( \text{OFF} \)

**Affected by Flags:** None

**Remarks:** When executed from a program, that program will resume execution when the calculator is turned on. This provides a programmable “autostart.”

**Related Commands:** CONT, HALT, KILL

---

**OLDPRT**

**Old Printer Command:** Modifies the remapping string in the reserved variable \( \text{PRTPAR} \) so that the extended character set of the HP 48 matches that of the HP 82240A Infrared Printer.

**Keyboard Access:** \( \leftarrow \) \( \text{I/O} \) PRINT PRTPAR OLDPR

**Affected by Flags:** None

**Remarks:** The character set in the HP 82240A Infrared Printer does not match the HP 48 character set:

- 24 characters in the HP 48 character set are not available in the HP 82240A Infrared Printer. (From the table in the keyword listing for NUM, these characters are numbers 129, 130, 143-157, 159, 166, 169, 172, 174, 184, and 185.) The HP 82240A prints a ¥ in substitution.
- Many characters in the extended character table (character codes 128 through 255) do not have the same character code. For example, the ¶ character has code 171 in the HP 48 and code 146 in the HP 82240A Infrared Printer.

To use the CHR command to print extended characters with an HP 82240A Infrared Printer, first execute OLDPR. The remapping string modified by OLDPR is the second parameter in \( \text{PRTPAR} \). This string (which is empty in the default state) changes the character code of each byte to match the codes in the HP 82240A Infrared Printer character table.
OLDPRT

To cancel OLDPRT character mapping, purge the variable PRTPAR, or enter \texttt{PRTPAR} \texttt{2 ` ^ PUT}.

To print a string containing graphics data, disable OLDPRT.

\textbf{Related Commands:} CR, DELAY, PRLCD, PRST, PRSTC, PRVAR, PR1

OPENIO

\textbf{Open I/O Port Command:} Opens the serial port or the IR port using the I/O parameters in the reserved variable \textit{IOPAR}.

\textbf{Keyboard Access:} \texttt{I/O \textbf{NXT SERIA OPENI}}

\textbf{Affected by Flags:} I/O Device (-33)

\textbf{Remarks:} Since all HP 48 Kermit-protocol commands automatically effect an OPENIO first, OPENIO is not normally needed, but can be used if an I/O transmission does not work. OPENIO is necessary for interaction with devices that interpret a closed port as a break.

OPENIO is also necessary for the automatic reception of data into the input buffer using non-Kermit commands. If the port is closed, incoming characters are ignored. If the port is open, incoming characters are automatically placed in the input buffer. These characters can be detected with BUFLEN, and can be read out of the input buffer using SRECV.

If the port is already open, OPENIO does not affect the data in the input buffer. However, if the port is closed, executing OPENIO clears the data in the input buffer.

For more information, refer to the reserved variable \textit{IOPAR} in appendix D, “Reserved Variables.”

\textbf{Related Commands:} BUFLEN, CLOSEIO, SBRK, SRECV, STIME, XMIT
OR Function: Returns the logical OR of two arguments.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>#n₂</td>
<td>→</td>
<td>#n₃</td>
</tr>
<tr>
<td>&quot;string₁&quot;</td>
<td>&quot;string₂&quot;</td>
<td>→</td>
<td>&quot;string₃&quot;</td>
</tr>
<tr>
<td>T/F₁</td>
<td>T/F₂</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>T/F</td>
<td>'symb'</td>
<td>→</td>
<td>'T/F OR symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>T/F</td>
<td>→</td>
<td>'symb OR T/F'</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>'symb₂'</td>
<td>→</td>
<td>'symb₁ OR symb₂'</td>
</tr>
</tbody>
</table>

Keyboard Access:

<table>
<thead>
<tr>
<th>MTH</th>
<th>BASE</th>
<th>NXT</th>
<th>LOGIC</th>
<th>OR</th>
</tr>
</thead>
</table>

| PRG | TEST | NXT | OR |

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: When the arguments are binary integers or strings, OR does a bit-by-bit (base 2) logical comparison.

- An argument that is a binary integer is treated as a sequence of bits as long as the current wordsize. Each bit in the result is determined by comparing the corresponding bits \((bit₁ \text{ and } bit₂)\) in the two arguments as shown in the following table.

<table>
<thead>
<tr>
<th>(bit₁)</th>
<th>(bit₂)</th>
<th>(bit₁ \text{ OR } bit₂)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- An argument that is a string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must be the same length.
OR

When the arguments are real numbers or symbolics, OR simply does a true/false test. The result is 1 (true) if either or both arguments are nonzero; it is 0 (false) if both arguments are zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic objects, then the result is an algebraic of the form \( \text{symb}_1 \text{ OR symb}_2 \). Execute \( \Rightarrow \text{NUM} \) (or set flag -3 before executing OR) to produce a numeric result from the algebraic result.

**Related Commands:** AND, NOT, XOR

ORDER

**Order Variables Command:** Reorders the variables in the current directory (shown in the VAR menu) to the order specified.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ global_1, ... global_n }</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** (MEMORY) [11FE]

**Affected by Flags:** None

**Remarks:** The names that appear first in the list will be the first to appear in the VAR menu. Variables not specified in the list are placed after the reordered variables.

If the list includes the name of a large subdirectory, there may be insufficient memory to execute ORDER.

**Related Commands:** VARS
**OVER**

**Over Command:** Returns a copy to stack level 1 of the object in level 2.

- **Level 2** | **Level 1** | **Level 3** | **Level 2** | **Level 1**
  - obj<sub>1</sub> | obj<sub>2</sub> | → | obj<sub>1</sub> | obj<sub>2</sub> | obj<sub>1</sub>

**Keyboard Access:** \( \text{STACK} \) \( \text{OVER} \)

**Affected by Flags:** None

**Related Commands:** PICK, ROLL, ROLLD, ROT, SWAP

---

**PARAMETRIC**

**Parametric Plot Type Command:** Sets the plot type to PARAMETRIC.

**Keyboard Access:** \( \text{PLOT} \) \( \text{PTYPE} \) \( \text{PARAMETRIC} \)

**Affected by Flags:** Simultaneous Plotting (–28), Curve Filling (–28)

**Remarks:** When the plot type is PARAMETRIC, the DRAW command plots the current equation as a complex-valued function of one real variable. The current equation is specified in the reserved variable **EQ**. The plotting parameters are specified in the reserved variable **PPAR**, which has the following form:

\[
\langle (x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}}) \rangle \text{ indep res axes ptype depend} \]

For plot type PARAMETRIC, the elements of **PPAR** are used as follows:

- \( (x_{\text{min}}, y_{\text{min}}) \) is a complex number specifying the lower left corner of **PICT** (the lower left corner of the display range). The default value is \((-5.5, -3.1)\).
PARAMETRIC

- \( (x_{\text{max}}, y_{\text{max}}) \) is a complex number specifying the upper right corner of \( PICT \) (the upper right corner of the display range). The default value is \( (6.5, 3.2) \).

- \( \text{indep} \) is a list containing a name that specifies the independent variable, and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). Note that the default value is \( X \). If \( X \) is not modified and included in a list with a plotting range, the values in \( (x_{\text{min}}, y_{\text{min}}) \) and \( (x_{\text{max}}, y_{\text{max}}) \) are used as the plotting range, which generally leads to meaningless results.

- \( \text{res} \) is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is 0, which specifies an interval equal to 1/130 of the difference between the maximum and minimum values in \( \text{indep} \) (the plotting range).

- \( \text{axes} \) is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is \( (0, 0) \).

- \( \text{ptype} \) is a command name specifying the plot type. Executing the command PARAMETRIC places the name PARAMETRIC in \( PPAR \).

- \( \text{depend} \) is a name specifying a label for the vertical axis. The default value is \( Y \).

The contents of \( EQ \) must be an expression or program; it cannot be an equation. It is evaluated for each value of the independent variable. The results, which must be complex numbers, give the coordinates of the points to be plotted. Lines are drawn between plotted points unless flag -31 is set.

If flag -28 is set, all equations are plotted simultaneously.

See chapter 23 of the \( HP 48 \ User's Guide \) for an example using the PARAMETRIC plot type.

**Related Commands:** BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE
PARITY

Parity Command: Sets the parity value in the reserved variable IOPAR.

<table>
<thead>
<tr>
<th>n-parity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

For more information, refer to the reserved variable IOPAR (I/O parameters) in appendix D, “Reserved Variables.”

Related Commands: BAUD, CKSM, TRANSIO
PARSURFACE

**PARSURFACE Plot Type Command:** Sets plot type to PARSURFACE.

**Keyboard Access:** (PLOT) (NXT) 3D PTYPE PARSU

**Affected by Flags:** None

**Remarks:** When plot type is set to PARSURFACE, the DRAW command plots an image graph of a 3-vector-valued function of two variables. PARSURFACE requires values in the reserved variables EQ, VPAR, and PPAR.

VPAR is made up of the following elements:

\[
\{ x_{\text{left}}, x_{\text{right}}, y_{\text{near}}, y_{\text{far}}, z_{\text{low}}, z_{\text{high}}, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}}, x_{\text{step}}, y_{\text{step}} \}
\]

For plot type PARSURFACE, the elements of VPAR are used as follows:

- \(x_{\text{left}}\) and \(x_{\text{right}}\) are real numbers that specify the width of the view space.
- \(y_{\text{near}}\) and \(y_{\text{far}}\) are real numbers that specify the depth of the view space.
- \(z_{\text{low}}\) and \(z_{\text{high}}\) are real numbers that specify the height of the view space.
- \(x_{\text{min}}\) and \(x_{\text{max}}\) are real numbers the specify the input region’s width. The default value is \((-1, 1)\).
- \(y_{\text{min}}\) and \(y_{\text{max}}\) are real numbers that specify the input region’s depth. The default value is \((-1, 1)\).
- \(x_{\text{eye}}, y_{\text{eye}},\) and \(z_{\text{eye}}\) are real numbers that specify the point in space from which the graph is viewed.
- \(x_{\text{step}}\) and \(y_{\text{step}}\) are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable PPAR, which has this form:

\[
\{ (x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}}), \text{indep res axes ptype depend} \}
\]
For plot type PARSURFACE, the elements of $PPAR$ are used as follows:

- $\langle x_{\text{min}}, y_{\text{min}} \rangle$ is not used.
- $\langle x_{\text{max}}, y_{\text{max}} \rangle$ is not used.
- $\text{indep}$ is a name specifying the independent variable. The default value of $\text{indep}$ is $X$.
- $\text{res}$ is not used.
- $\text{axes}$ is not used.
- $\text{ptype}$ is a command name specifying the plot type. Executing the command PARSURFACE places the name PARSURFACE in $\text{ptype}$.
- $\text{depend}$ is a name specifying the dependent variable. The default value is $Y$.

**Related Commands:** BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

---

**PATH**

**Current Path Command:** Returns a list specifying the path to the current directory.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow$</td>
<td>$\rightarrow { \text{HOME} \ directory-name_1 \ldots \ directory-name_n }$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{MEMORY} \ \text{DIR} \ \text{PATH}\)

**Affected by Flags:** None

**Remarks:** The first directory is always $HOME$, and the last directory is always the current directory.

If a program needs to switch to a specific directory, it can do so by evaluating a directory list, such as one created earlier by PATH.
Related Commands: CRDIR, HOME, PGDIR, UPDIR

PCOEF

Monic Polynomial Coefficients Command: Returns the coefficients of a monic polynomial (a polynomial with a leading coefficient of 1) having specific roots.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ array ]_roots</td>
<td>→</td>
<td>[ array ]_coefficients</td>
</tr>
</tbody>
</table>

Keyboard Access: («)(SOLVE) POLY PCOEF

Affected by Flags: None

Remarks: The argument must be a real or complex array of length n containing the polynomial's roots. The result is a real or complex vector of length n+1 containing the coefficients listed from highest order to lowest, with a leading coefficient of 1.

Example: Find the polynomial that has the roots 2, -3, 4, -5:

```
[ 2 -3 4 -5 ] PCOEF returns [ 1 2 -25 -26 120 ], representing the polynomial \( x^4 + 2x^3 - 25x^2 - 26x + 120. \)
```

Related Commands: PEVAL, PROOT
PCONTOUR

PCONTOUR Plot Type Command: Sets the plot type to PCONTOUR.

Keyboard Access:  

Affected by Flags: None

Remarks: When plot type is set PCONTOUR, the DRAW command plots a contour-map view of a scalar function of two variables. PCONTOUR requires values in the reserved variables \( EQ, VPAR, \) and \( PPAR. \)

\( VPAR \) is made up of the following elements:

\[ \{ x_{\text{left}}, x_{\text{right}}, y_{\text{near}}, y_{\text{far}}, z_{\text{low}}, z_{\text{high}}, x_{\min}, x_{\max}, y_{\min}, y_{\max}, x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}}, x_{\text{step}}, y_{\text{step}} \} \]

For plot type PCONTOUR, the elements of \( VPAR \) are used as follows:

- \( x_{\text{left}} \) and \( x_{\text{right}} \) are real numbers that specify the width of the view space.
- \( y_{\text{near}} \) and \( y_{\text{far}} \) are real numbers that specify the depth of the view space.
- \( z_{\text{low}} \) and \( z_{\text{high}} \) are real numbers that specify the height of the view space.
- \( x_{\min} \) and \( x_{\max} \) are not used.
- \( y_{\min} \) and \( y_{\max} \) are not used.
- \( x_{\text{eye}}, y_{\text{eye}}, \) and \( z_{\text{eye}} \) are real numbers that specify the point in space from which the graph is viewed.
- \( x_{\text{step}} \) and \( y_{\text{step}} \) are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable \( PPAR, \) which has this form:

\[ \{ (x_{\min}, y_{\min}), (x_{\max}, y_{\max}), \text{indep} \text{ res} \text{ axes} \text{ ptype} \text{ depend} \} \]

For plot type PCONTOUR, the elements of \( PPAR \) are used as follows:

- \( (x_{\min}, y_{\min}) \) is not used.
- \( (x_{\max}, y_{\max}) \) is not used.


PCONTOUR

- **indep** is a name specifying the independent variable. The default value of **indep** is X.
- **res** is not used.
- **axes** is not used.
- **ptype** is a command name specifying the plot type. Executing the command PCONTOUR places the name PCONTOUR in *ptype*.
- **depend** is a name specifying the dependent variable. The default value is Y.

**Related Commands:** BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PCOV

**Population Covariance Command:** Returns the population covariance of the independent and dependent data columns in the current statistics matrix (reserved variable $\Sigma\, DAT$).

- **Level 1**
  - $\rightarrow$
  - $\rightarrow x_{pcovariance}$

**Keyboard Access:** $\leftarrow$ STAT FIT NXT PCOV

**Affected by Flags:** None

**Remarks:** The columns are specified by the first two elements in reserved variable $\Sigma\, PAR$, set by XCOL and YCOL respectively. If $\Sigma\, PAR$ does not exist, PCOV creates it and sets the elements to their default values (1 and 2).

The population covariance is calculated with the following formula:

$$
\frac{1}{n} \sum_{k=1}^{n} (x_{kn_1} - \bar{x}_{n_1})(x_{kn_2} - \bar{x}_{n_2})
$$
where $x_{kn_1}$ is the $k$th coordinate value in column $n_1$, $x_{kn_2}$ is the $k$th coordinate value in the column $n_2$, $\overline{x_{n_1}}$ is the mean of the data in column $n_1$, $\overline{x_{n_2}}$ is the mean of the data in column $n_2$, and $n$ is the number of data points.

**Related Commands:** COLΣ, CORR, COV, PREDX, PREDY, XCOL, YCOL

---

**PDIM**

**PICT Dimension Command:** Replaces PICT with a blank PICT of the specified dimensions.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x_{\text{min}}, y_{\text{min}})$</td>
<td>$(x_{\text{max}}, y_{\text{max}})$</td>
<td>→</td>
<td>$(x_{\text{max}}, y_{\text{max}})$</td>
</tr>
<tr>
<td>$#n_{\text{width}}$</td>
<td>$#m_{\text{height}}$</td>
<td>→</td>
<td>$#m_{\text{height}}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** PICT PDIM

**Affected by Flags:** None

**Remarks:** If the arguments are complex numbers, PDIM changes the size of PICT and makes the arguments the new values of $(x_{\text{min}}, y_{\text{min}})$ and $(x_{\text{max}}, y_{\text{max}})$ in the reserved variable $PPAR$. Thus, the scale of a subsequent plot is not changed. If the arguments are binary integers, $PPAR$ remains unchanged, so the scale of a subsequent plot is changed.

PICT cannot be smaller than 131 pixels wide $\times$ 64 pixels high, nor wider than 2048 pixels (height is unlimited).

**Related Commands:** PMAX, PMIN

---

Command Reference  3-223
PERM

Permutations Function: Returns the number of possible permutations of \( n \) items taken \( m \) at a time.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( m )</td>
<td>( \rightarrow )</td>
<td>( P_{n,m} )</td>
</tr>
<tr>
<td>'symb(_n)'</td>
<td>( m )</td>
<td>( \rightarrow )</td>
<td>'PERM(symb(_n), m)'</td>
</tr>
<tr>
<td>( n )</td>
<td>'symb(_m)'</td>
<td>( \rightarrow )</td>
<td>'PERM(n, symb(_m))'</td>
</tr>
<tr>
<td>'symb(_n)'</td>
<td>'symb(_m)'</td>
<td>( \rightarrow )</td>
<td>'PERM(symb(_n), symb(_m))'</td>
</tr>
</tbody>
</table>

Keyboard Access: \[ \text{MTH} \text{ NXT} \text{ PROB} \text{ PERM} \]

Affected by Flags: Numerical Results (−3)

Remarks: The formula used to calculate \( P_{n,m} \) is this:

\[
P_{n,m} = \frac{n!}{(n - m)!}
\]

The arguments \( n \) and \( m \) must each be less than \( 10^{12} \).

Related Commands: COMB, !

PEVAL

Polynomial Evaluation Command: Evaluates an \( n \)-degree polynomial at \( x \).

\[
\begin{array}{|c|c|c|}
\hline
\text{Level 2} & \text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
[\text{array}]_{\text{coefficients}} & x & \rightarrow & p(x) \\
\hline
\end{array}
\]

Keyboard Access: \[ \text{SOLVE} \text{ POLY} \text{ PEVAL} \]

3-224 Command Reference
Affected by Flags: None

Remarks: The arguments must be an array of length \( n+1 \) containing the polynomial’s coefficients listed from highest order to lowest, and the value \( x \) at which the polynomial is to be evaluated.

Example: Evaluate the polynomial \( x^4 + 2x^3 - 25x^2 - 26x + 120 \) at \( x = 8 \):

\[ [1, 2, -25, -26, 120] \cdot 8 \] returns 3432.

Related Commands: PCOEF, PROOT

PGDIR

Purge Directory Command: Purges the named directory (whether empty or not).

```
{}
```

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'global'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( \mathbf{\uparrow} \) MEMORY DIR PGDIR

Affected by Flags: None

Related Commands: CLVAR, CRDIR, HOME, PATH, PURGE, UPDIR
**PICK**

**Pick Object Command:** Copies the contents of a specified level to level 1.

\[
\begin{array}{cccc}
\text{Level } n+1.. & \text{Level } 2 & \text{Level } 1 & \rightarrow & \text{Level } n+1.. & \text{Level } 2 & \text{Level } 1 \\
obj_n & \ldots & obj_1 & n & \rightarrow & obj_n & \ldots & obj_1 & obj_n \\
\end{array}
\]

**Keyboard Access:** \( \text{\{\text{STACK}\} \text{ PICK} \) \text{ PICK} \)

**Affected by Flags:** None

**Related Commands:** DUP, DUPN, DUP2, OVER, ROLL, ROLLD, ROT, SWAP

---

**PICT**

**PICT Command:** Puts the name PICT on the stack.

\[
\begin{array}{ccc}
\text{Level } 1 & \rightarrow & \text{Level } 1 \\
& \rightarrow & \text{PICT} \\
\end{array}
\]

**Keyboard Access:** \( \text{\{PRG\} PICT} PICT \)

**Affected by Flags:** None

**Remarks:** PICT is the name of a storage location in calculator memory containing the current graphics object. The command PICT enables access to the contents of that memory location as if it were a variable. Note, however, that PICT is not a variable as defined in the HP 48: its name cannot be quoted, and only graphics objects may be "stored" in it.
If a graphics object smaller than 131 wide x 64 pixels high is stored in PICT, it is enlarged to 131 x 64. A graphics object of unlimited pixel height and up to 2048 pixels wide can be stored in PICT.

**Examples:** PICT RCL returns the current graphics object to the stack.

GRAPHIC 131 × 64 PICT STO stores a graphics object in PICT, making it the current graphics object.

**Related Commands:** GOR, GXOR, NEG, PICTURE, PVIEW, RCL, REPL, SIZE, STO, SUB

---

**PICTURE**

**Picture Environment Command:** Selects the Picture environment (selects the graphics display and activates the graphics cursor and Picture menu).

**Keyboard Access:** $\text{EXIT} \text{PICTURE}$

**Affected by Flags:** None

**Remarks:** When executed from a program, PICTURE suspends program execution until $\text{CANCEL}$ is pressed.

**Example:** This program:

```plaintext
"Press CANCEL to return to stack" 1 DISP 3 WAIT PICTURE
```

displays a message for 3 seconds, then selects the Picture environment. (The $\text{m}$ character in the program indicates a linefeed.)

**Related Commands:** PICTURE, PVIEW, TEXT
**PINIT**

**Port Initialize Command:** Initializes all currently active ports. Does not affect data already stored in a port.

**Keyboard Access:** ←LIBRARY NXT PINIT

**Affected by Flags:** None

**Related Commands:** None

**Remarks:** PINIT is particularly useful when using a plug-in card that can hold multiple ports. It stores and then purges an object in each port (128K partition) that can be accessed at the time the command is executed. This has the effect of initializing each port without disturbing any previously-stored data.

---

**PIXOFF**

**Pixel Off Command:** Turns off the pixel at the specified coordinate in *PICT*.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x, y))</td>
<td>→</td>
<td>({ #n #m })</td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG PICT NXT PIXOFF

**Affected by Flags:** None

**Related Commands:** PIXON, PIX?
**PIXON**

**Pixel On Command:**
Turns on the pixel at the specified coordinate in *PICT*.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x, \ y))</td>
<td>→</td>
<td>(n)</td>
</tr>
<tr>
<td>({ #n #m } )</td>
<td>→</td>
<td>(m)</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{PRG} \ \text{PICT} \ \text{NXT} \ \text{PIXON} \)

**Affected by Flags:** None

**Related Commands:** PIXOFF, PIX?

---

**PIX?**

**Pixel On? Command:**
Tests whether the specified pixel in *PICT* is on; returns 1 (true) if the pixel is on, and 0 (false) if the pixel is off.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x, \ y))</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>({ #n #m } )</td>
<td>→</td>
<td>0/1</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{PRG} \ \text{PICT} \ \text{NXT} \ \text{PIX?} \)

**Affected by Flags:** None

**Related Commands:** PIXON, PIXOFF
**PKT**

**Packet Command:** Used to send command “packets” (and receive requested data) to a Kermit server.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;data&quot;</td>
<td>&quot;type&quot;</td>
<td>→</td>
<td>&quot;response&quot;</td>
</tr>
</tbody>
</table>

**Keyboard Access:** 

**Affected by Flags:** I/O Device (−33), I/O Messages (−39)

The I/O Data Format flag (−35) can be significant if the server sends back more than one packet.

**Remarks:** To send HP 48 objects, use SEND.

PKT allows additional commands to be sent to a Kermit server. For more information, refer to *Using MS-DOS Kermit* by Christine M. Gianone, Digital Press, 1990; or *KERMIT, A File Transfer Protocol* by Frank da Cruz, Digital Press, 1987, especially chapter 11, “The Client/Server Model.”

The packet data, packet type, and the response to the packet transmission are all in string form. PKT first does an I (initialization) packet exchange with the Kermit server, then sends the server a packet constructed from the data and packet-type arguments supplied to PKT. The response to PKT will be either an acknowledging string (possibly blank) or an error packet (see KERRM).

For the *type* argument, only the first letter is significant.

**Examples:** A PKT command to send a generic directory request is "D" "G" PKT.

To send a *host command* packet, use a command from the server’s operating system for the *data* string and "C" for the *type* string. For example, "'ABC' PURGE" "C" PKT on a local HP 48 would instruct an HP 48 server to purge variable ABC.

**Related Commands:** CLOSEIO, KERRM, SERVER
Pmax

PICT Maximum Command: Specifies \((x, y)\) as the coordinates at the upper right corner of the display.

\[
\begin{array}{c|c}
\text{Level 1} & \text{Level 1} \\
(x, y) & \rightarrow \\
\end{array}
\]

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: The complex number \((x, y)\) is stored as the second element in the reserved variable \(PPAR\).

Related Commands: PDIM, PMIN, XRNG, YRNG

Pmin

PICT Minimum Command: Specifies \((x, y)\) as the coordinates at the lower left corner of the display.

\[
\begin{array}{c|c}
\text{Level 1} & \text{Level 1} \\
(x, y) & \rightarrow \\
\end{array}
\]

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: The complex number \((x, y)\) is stored as the first element in the reserved variable \(PPAR\).

Related Commands: PDIM, PMAX, XRNG, YRNG
POLAR

Polar Plot Type Command: Sets the plot type to POLAR.

Keyboard Access: (¢)(PLOT) PTYPE POLAR

Affected by Flags: Simultaneous Plotting (–28), Curve Filling (–31)

Remarks: When the plot type is POLAR, the DRAW command plots the current equation in polar coordinates, where the independent variable is the polar angle and the dependent variable is the radius. The current equation is specified in the reserved variable EQ.

The plotting parameters are specified in the reserved variable PPAR, which has this form:

\[ \langle x_{\min}, y_{\min} \rangle \langle x_{\max}, y_{\max} \rangle \text{ indep res axes ptype depend} \]

For plot type POLAR, the elements of PPAR are used as follows:

- \( \langle x_{\min}, y_{\min} \rangle \) is a complex number specifying the lower left corner of PICT (the lower left corner of the display range). The default value is \((–6.5, –3.1)\).

- \( \langle x_{\max}, y_{\max} \rangle \) is a complex number specifying the upper right corner of PICT (the upper right corner of the display range). The default value is \((6.5, 3.2)\).

- \( \text{indep} \) is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of \( \text{indep} \) is \( \text{X} \).

- \( \text{res} \) is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is \( \emptyset \), which specifies an interval of 2 degrees, 2 grads, or \( \pi/90 \) radians.

- \( \text{axes} \) is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is \( \langle \emptyset, \emptyset \rangle \).

- \( \text{ptype} \) is a command name specifying the plot type. Executing the command POLAR places the name POLAR in \( \text{ptype} \).
- depend is a name specifying a label for the vertical axis. The default value is $Y$.

The current equation is plotted as a function of the variable specified in indep. The minimum and maximum values of the independent variable (the plotting range) can be specified in indep; otherwise, the default minimum value is 0 and the default maximum value corresponds to one full circle in the current angle mode (360 degrees, 400 grads, or $2\pi$ radians). Lines are drawn between plotted points unless flag $-31$ is set.

If flag $-28$ is set, all equations are plotted simultaneously.

If $EQ$ contains an expression or program, the expression or program is evaluated in Numerical Results mode for each value of the independent variable to give the values of the dependent variable. If $EQ$ contains an equation, the plotting action depends on the form of the equation.

<table>
<thead>
<tr>
<th>Form of Current Equation</th>
<th>Plotting Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>'expr=expr'</td>
<td>Each expression is plotted separately. The intersection of the two graphs shows where the expressions are equal.</td>
</tr>
<tr>
<td>'name=expr'</td>
<td>Only the expression is plotted.</td>
</tr>
</tbody>
</table>

Related Commands: BAR, CONIC, DIFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE
POS

**Position Command:** Returns the position of a substring within a string or the position of an object within a list.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;string&quot;</td>
<td>&quot;substring&quot;</td>
<td>→</td>
<td>n</td>
</tr>
<tr>
<td>{ list }</td>
<td>obj</td>
<td>→</td>
<td>n</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- Left Bracket + CHAR + POS
- PRG + LIST + ELEM + POS

**Affected by Flags:** None

**Remarks:** If there is no match for `obj` or `substring`, POS returns zero.

**Related Commands:** CHR, NUM, REPL, SIZE, SUB

PREDV

**Predicted y-Value Command:** Returns the predicted dependent-variable value \( y_{\text{dependent}} \), based on the independent-variable value \( x_{\text{independent}} \), the currently selected statistical model, and the current regression coefficients in the reserved variable \( \Sigma PAR \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{\text{independent}} )</td>
<td>→</td>
<td>( y_{\text{dependent}} )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** None. Must be typed in.
Remarks: Provided for compatibility with the HP 28. PREDV is the same as PREDY. See PREDY.

PREDX

Predicted x-Value Command: Returns the predicted independent-variable value $x_{\text{independent}}$, based on the dependent-variable value $y_{\text{dependent}}$, the currently selected statistical model, and the current regression coefficients in the reserved variable $\Sigma PAR$.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{\text{dependent}}$</td>
<td>→</td>
<td>$x_{\text{independent}}$</td>
</tr>
</tbody>
</table>

Keyboard Access: $(\mathbf{STAT}) F1 F4 PREDX$

Affected by Flags: None

Remarks: The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable $\Sigma PAR$. For the linear statistical model, the equation used is this:

$$y_{\text{dependent}} = (mx_{\text{independent}}) + b$$

where $m$ is the slope (the third element in $\Sigma PAR$) and $b$ is the intercept (the fourth element in $\Sigma PAR$).

For the other statistical models, the equations used by PREDX are listed in the LR entry.

If PREDX is executed without having previously generated regression coefficients in $\Sigma PAR$, a default value of zero is used for both regression coefficients, and an error results.

Example: Given five columns of data in $\Sigma DAT$, the command sequence:

```
2 XCOL 5 YCOL LOGFIT LR 23 PREDX
```

sets column 2 as the independent variable column, sets column 5 as the dependent variable column, and sets the logarithmic statistical
PREDX

model. It then executes LR, generating intercept and slope regression coefficients, and storing them in $\Sigma PAR$. Then, given a dependent value of $Z$, it returns a predicted independent value based on the regression coefficients and the statistical model.

**Related Commands:** COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, LR, PREDY, PWRFIT, XCOL, YCOL

PREDY

**Predicted y-Value Command:** Returns the predicted dependent-variable value $y_{dependent}$, based on the independent-variable value $x_{independent}$, the currently selected statistical model, and the current regression coefficients in the reserved variable $\Sigma PAR$.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{independent}$</td>
<td>→</td>
<td>$y_{dependent}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (q)(STAT)Fit PREDY

**Affected by Flags:** None

**Remarks:** The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable $\Sigma PAR$. For the linear statistical model, the equation used is this:

$$y_{dependent} = (m x_{independent}) + b$$

where $m$ is the slope (the third element in $\Sigma PAR$) and $b$ is the intercept (the fourth element in $\Sigma PAR$).

For the other statistical models, the equations used by PREDY are listed in the LR entry.

If PREDY is executed without having previously generated regression coefficients in $\Sigma PAR$, a default value of zero is used for both regression coefficients—in this case PREDY will return $\emptyset$ for statistical models LINFIT and LOGFIT, and error for statistical models EXPFIT and PWRFIT.
Example: Given four columns of data in \( \Sigma \text{DAT} \), the command sequence:

\[
\text{2 XCOL 4 YCOL PWRFIT LR 11 PREDY}
\]

sets column 2 as the independent variable column, sets column 4 as the dependent variable column, and sets the power statistical model. It then executes LR, generating intercept and slope regression coefficients, and storing them in \( \Sigma \text{PAR} \). Then, given an independent value of 11, it returns a predicted dependent value based on the regression coefficients and the statistical model.

Related Commands: \text{COL2, CORR, COV, EXPFIT, SLINE, LINFIT, LOGFIT, LR, PREDX, PWRFIT, XCOL, YCOL}

---

PRLCD

Print LCD Command: Prints a pixel-by-pixel image of the current display (excluding the annunciators).

Keyboard Access: \( \text{\#} \text{I/O PRNT PRLCD} \)

Affected by Flags: Printing Device \((-34)\), I/O Device \((-33)\), Linefeed \((-38)\)

If flag \(-34\) is set (printer output directed to the serial port), flag \(-33\) must be clear.

Flag \(-38\) must be clear.

Remarks: The width of the printed image of characters in the display is narrower using PRLCD than using a print command such as PR1. The difference results from the spacing between characters. On the display there is a single blank column between characters, and PRLCD prints this spacing. Print commands such as PR1 print two blank columns between adjacent characters.

Example: The command sequence ERASE DRAW PRLCD clears \text{PICT}, plots the current equation, then prints the graphics display.

Related Commands: \text{CR, DELAY, OLDPRT, PRST, PRSTC, PRVAR, PR1}
PROMPT

Prompt Command: Displays the contents of "prompt" in the status area, and halts program execution.

{}  

Level 1  →  Level 1

"prompt"  →  

Keyboard Access: PRG NXT IN NXT PROM

Affected by Flags: None

Remarks: PROMPT is equivalent to 1 DISP 1 FREEZE HALT.

Related Commands: CONT, DISP, FREEZE, HALT, INFORM, INPUT, MSGBOX

PROOT

Polynomial Roots Command: Returns all roots of an n-degree polynomial having real or complex coefficients.

{}  

Level 1  →  Level 1

[ array ]coefficients  →  [ array ]roots

Keyboard Access: SOLVE POLY PROOT

Affected by Flags: Infinite Result Exception (−22)

Remarks: For an n°-order polynomial, the argument must be a real or complex array of length n+1 containing the coefficients listed from highest order to lowest. The result is a real or complex vector of length n containing the computed roots.
PROOT interprets leading coefficients of zero in a limiting sense. As a leading coefficient approaches zero, a root of the polynomial approaches infinity: therefore, if flag —22 is clear (the default), PROOT reports an Infinite Result error if a leading coefficient is zero. If flag —22 is set, PROOT returns a root of (MAXREAL,0) for each leading zero in an array containing real coefficients, and a root of (MAXREAL,M铣XREAL) for each leading zero in an array containing complex coefficients.

Example: Find the roots of the polynomial \( x^4+2x^3-25x^2-26x+120 \):
\[
\begin{bmatrix}
1 & 2 & -25 & -26 & 120
\end{bmatrix}
\]
PROOT returns 
\[
\begin{bmatrix}
2 & -3 & 4 & -5
\end{bmatrix}
\]

Related Commands: PCOEF, PEVAL

---

**PRST**

**Print Stack Command:** Prints all objects in the stack, starting with the object in the highest level.

**Keyboard Access:** \( \text{[→]}\) I/O PRINT PRST

**Affected by Flags:** Double-Spaced Printing (—37), Printing Device (—34), I/O Device (—33), Linefeed (—38)

If flag —34 is set (printer output directed to the serial port), flag —33 must be clear.

When flag —38 is set, linefeeds are not added at the end of each print line. Generally, flag —38 should be clear for execution of PRST. PRST leaves the stack unchanged.

**Remarks:** Objects are printed in multiline printer format. See the PR1 entry for a description of multiline printer format.

**Related Commands:** CR, DELAY, OLDPRT, PRLCD, PRSTC, PRVAR, PR1
PRSTC
Print Stack (Compact) Command: Prints in compact form all objects in the stack, starting with the object in the highest level.
Keyboard Access: ➪I/O PRINT PRSTC
Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33), Linefeed (−38)
If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.
When flag −38 is set, linefeeds are not added at the end of each print line. Generally, flag −38 should be clear for execution of PRSTC.
Remarks: Compact printer format is the same as compact display format. Multiline objects are truncated and appear on one line only. PRSTC leaves the stack unchanged.
Related Commands: CR, DELAY, OLDPRT, PRLCD, PRST, PRVAR, PR1

PRVAR
Print Variable Command: Searches the current directory path or port for the specified variables and prints the name and contents of each variable.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ name₁, name₂, ... }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>:n₉-port :'global'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: ➪I/O PRINT PRVAR
Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33), Linefeed (−38)
PR1

Print Level 1 Command: Prints an object in multiline printer format.

Keyboard Access: \( \text{\textasciitilde I/O } PR1 \)

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33), Linefeed (−38)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

Remarks: All objects except strings are printed with their identifying delimiters. Strings are printed without the leading and trailing "delimiters. PR1 leaves the stack unchanged.

Multiline printer format is similar to multilinedisplay format, with the following exceptions:

- Strings and names that are more than 24 characters long are continued on the next printer line.
- The real and imaginary parts of complex numbers are printed on separate lines if they don’t fit on the same line.
- Arrays are printed with a numbered heading for each row and with a column number before each element. For example, the 2 × 3 array

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\]

would be printed as follows:
Array (2 3) — Array dimensions

Row number — Row 1
  1] 1

Column number
  2] 2
  3] 3

Row 2
  1] 4
  2] 5
  3] 6

Related Commands: CR, DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR

PSDEV

Population Standard Deviation Command: Calculates the population standard deviation of each of the m columns of coordinate values in the current statistics matrix (reserved variable \( \Sigma DAT \)).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>( x_{psdev} )</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>( [x_{psdev1}, x_{psdev2}, \ldots, x_{psdevm}] )</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{STAT} \text{IVAR} \text{NXT} \text{PSDEV} \)

Affected by Flags: None

Remarks: PSDEV returns a vector of \( m \) real numbers, or a single real number if \( m = 1 \). The population standard deviation is computed using this formula:

3-242 Command Reference
where $x_k$ is the $k$th coordinate value in a column, $\bar{x}$ is the mean of the data in this column, and $n$ is the number of data points.

**Related Commands:** MEAN, PCOV, PVAR, SDEV, TOT, VAR

---

**PURGE**

**Purge Command:** Purges the named variables or empty subdirectories from the current directory.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'global'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ $global_1, \ldots, global_n$ }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>PICT</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>:$n_{port}$ :name$_{backup}$</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>:$n_{port}$ :n$_{library}$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** 📧 PURGE

**Affected by Flags:** None

**Remarks:** PURGE executed in a program does not save its argument for recovery by LASTARG.

To empty a named directory before purging it, use PGDIR.

To help prepare a list of variables for purging, use VARS.

Purging PICT replaces the current graphics object with a $0 \times 0$ graphics object.

If a list of objects (with global names, backup objects, library objects, or PICT) for purging contains an invalid object, then the objects...
PURGE

preceding the invalid object are purged, and the error Bad Argument Type occurs.

To purge a library or backup object, tag the library number or backup name with the appropriate port number (*n_port), which must be in the range from 0 to 33. (A library can be purged from RAM only.) For a backup object, the port number can be replaced with the wildcard character %, in which case the HP 48 will search ports 33 through 0, and then main memory for the named backup object.

Library objects in RAM can be purged, while those in ROM (application cards and write-protected RAM cards) cannot. A library object must be detached before it can be purged from the HOME directory.

Neither a library object nor a backup object can be purged if it is currently “referenced” internally by stack pointers (such as an object on the stack, in a local variable, on the LAST stack, or on an internal return stack). This produces the error Object in Use. To avoid these restrictions, use NEWOB before purging. (See NEWOB.)

Related Commands: CLEAR, CLUSR, CLVAR, NEWOB, PGDIR

PUT

Put Element Command: In the level 3 array or list, PUT replaces with \( z_{\text{put}} \) or \( obj_{\text{put}} \) the object whose position is specified in level 2; if the array or list is unnamed, returns the new array or list.
### PUT

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[ matrix ]]</td>
<td>( n_{\text{position}} )</td>
<td>( z_{\text{put}} )</td>
<td>→</td>
<td>[[ matrix ]]</td>
</tr>
<tr>
<td>[[ matrix ]]</td>
<td>{ ( n_{\text{row}} ) ( m_{\text{col}} ) }</td>
<td>( z_{\text{put}} )</td>
<td>→</td>
<td>[[ matrix ]]</td>
</tr>
<tr>
<td>'name' matrix</td>
<td>( n_{\text{position}} )</td>
<td>( z_{\text{put}} )</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>'name' matrix</td>
<td>{ ( n_{\text{row}} ) ( m_{\text{col}} ) }</td>
<td>( z_{\text{put}} )</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>[ vector ]</td>
<td>( n_{\text{position}} )</td>
<td>( z_{\text{put}} )</td>
<td>→</td>
<td>[ vector ]</td>
</tr>
<tr>
<td>[ vector ]</td>
<td>{ ( n_{\text{position}} ) }</td>
<td>( z_{\text{put}} )</td>
<td>→</td>
<td>[ vector ]</td>
</tr>
<tr>
<td>'name' vector</td>
<td>( n_{\text{position}} )</td>
<td>( z_{\text{put}} )</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>'name' vector</td>
<td>{ ( n_{\text{position}} ) }</td>
<td>( z_{\text{put}} )</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ list }</td>
<td>( n_{\text{position}} )</td>
<td>( \text{obj}_{\text{put}} )</td>
<td>→</td>
<td>{ list }</td>
</tr>
<tr>
<td>{ list }</td>
<td>{ ( n_{\text{position}} ) }</td>
<td>( \text{obj}_{\text{put}} )</td>
<td>→</td>
<td>{ list }</td>
</tr>
<tr>
<td>'name' list</td>
<td>( n_{\text{position}} )</td>
<td>( \text{obj}_{\text{put}} )</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>'name' list</td>
<td>{ ( n_{\text{position}} ) }</td>
<td>( \text{obj}_{\text{put}} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:**  \( \text{PRG LIST ELEM PUT} \)

**Affected by Flags:** None

**Remarks:** For matrices, \( n_{\text{position}} \) counts in row order.

If the argument in level 3 is a name, PUT alters the named array or list and returns nothing to the stack.

**Examples:** This command sequence:

\[
[[ 2 3 4 ] [ 4 1 2 ] ] [ 1 3 ] 96 \text{ PUT returns} \\
[[ 2 3 96 ] [ 4 1 2 ] ]
\]

The command sequence \( [[ 2 3 4 ] [ 4 1 2 ] ] 5 96 \) \text{ PUT returns} \( [[ 2 3 4 ] [ 4 96 2 ] ] \).

The command sequence \( \{ A B C D E \} \{ 3 \} 'Z' \) \text{ PUT returns} \( \{ A B Z D E \} \).

**Related Commands:** GET, GETI, PUTI
**PUTI**

**Put and Increment Index Command:** In the level 3 array or list, replaces with $z_{\text{put}}$ or $obj_{\text{put}}$ the object whose position is specified in level 2, returning the new array or list and the next position in that array or list.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[[\text{matrix}]]_1$</td>
<td>$n_{\text{pos}1}$</td>
<td>$z_{\text{put}}$</td>
<td>$[[\text{matrix}]]_2$</td>
<td>$n_{\text{pos}2}$</td>
</tr>
<tr>
<td>$[[\text{matrix}]]_2$</td>
<td>${n_r\ mc}_1$</td>
<td>$z_{\text{put}}$</td>
<td>$[[\text{matrix}]]_2$</td>
<td>${n_r\ mc}_2$</td>
</tr>
<tr>
<td>'name'$_{\text{matrix}}$</td>
<td>$n_{\text{pos}1}$</td>
<td>$z_{\text{put}}$</td>
<td>'name'$_{\text{matrix}}$</td>
<td>$n_{\text{pos}2}$</td>
</tr>
<tr>
<td>'name'$_{\text{matrix}}$</td>
<td>${n_r\ mc}_1$</td>
<td>$z_{\text{put}}$</td>
<td>'name'$_{\text{matrix}}$</td>
<td>${n_r\ mc}_2$</td>
</tr>
<tr>
<td>$[\text{vector}]_1$</td>
<td>$n_{\text{pos}1}$</td>
<td>$z_{\text{put}}$</td>
<td>$[\text{vector}]_2$</td>
<td>$n_{\text{pos}2}$</td>
</tr>
<tr>
<td>$[\text{vector}]_2$</td>
<td>${n_{\text{pos}1}}$</td>
<td>$z_{\text{put}}$</td>
<td>$[\text{vector}]_2$</td>
<td>${n_{\text{pos}2}}$</td>
</tr>
<tr>
<td>'name'$_{\text{vector}}$</td>
<td>$n_{\text{pos}}$</td>
<td>$z_{\text{put}}$</td>
<td>'name'$_{\text{vector}}$</td>
<td>$n_{\text{pos}2}$</td>
</tr>
<tr>
<td>{list}$_1$</td>
<td>$n_{\text{pos}1}$</td>
<td>$obj_{\text{put}}$</td>
<td>{list}$_2$</td>
<td>$n_{\text{pos}2}$</td>
</tr>
<tr>
<td>{list}$_2$</td>
<td>${n_{\text{pos}1}}$</td>
<td>$obj_{\text{put}}$</td>
<td>{list}$_2$</td>
<td>${n_{\text{pos}2}}$</td>
</tr>
<tr>
<td>'name'$_{\text{list}}$</td>
<td>$n_{\text{pos}1}$</td>
<td>$obj_{\text{put}}$</td>
<td>'name'$_{\text{list}}$</td>
<td>$n_{\text{pos}2}$</td>
</tr>
<tr>
<td>'name'$_{\text{list}}$</td>
<td>${n_{\text{pos}1}}$</td>
<td>$obj_{\text{put}}$</td>
<td>'name'$_{\text{list}}$</td>
<td>${n_{\text{pos}2}}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG LIST ELEM PUTI

**Affected by Flags:** Index Wrap Indicator (-64)

The Index Wrap Indicator flag is cleared on each execution of PUTI until the position (index) wraps to the first position in the array or list, at which point the flag is set. The next execution of PUTI again clears the flag.

**Remarks:** For matrices, the position is incremented in row order.

Unlike PUT, PUTI returns a named array or list (to level 2). This enables a subsequent execution of PUTI at the next position of a named array or list.

**Example:** The following program uses PUTI and flag -64 to replace $A$, $B$, and $C$ in the list with $X$.

3-246 Command Reference
PVAR

Population Variance Command: Calculates the population variance of the coordinate values in each of the \( m \) columns in the current statistics matrix (\( \sum DAT \)).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow )</td>
<td>→</td>
<td>( x_{pvariance} )</td>
</tr>
<tr>
<td>( \rightarrow )</td>
<td>→</td>
<td>( [x_{pvariance1} \ldots x_{pvariancem}] )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \leftarrow \text{STAT} \text{ VAR} \text{ NXT} \text{ PVAR} \)

Affected by Flags: None

Remarks: The population variance (equal to the square of the population standard deviation) is returned as a vector of \( m \) real numbers, or as a single real number if \( m = 1 \). The population variances are computed using this formula:

\[
\frac{1}{n} \sum_{k=1}^{n} (x_k - \bar{x})^2
\]

where \( x_k \) is the \( k \)th coordinate value in a column, \( \bar{x} \) is the mean of the data in this column, and \( n \) is the number of data points.

Related Commands: MEAN, PCOV, PSDEV, SDEV, VAR
**PVARS**

**Port-Variables Command:** Returns a list of the backup objects ($n_{\text{port}}$ $name$) and the library objects ($n_{\text{port}}$ $n_{\text{library}}$) in the specified port. Also returns the available memory size (if RAM) or the memory type.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{port}}$</td>
<td>${n_{\text{port}} : name_{\text{backup}} \ldots}$</td>
<td>memory</td>
</tr>
<tr>
<td>$n_{\text{port}}$</td>
<td>${n_{\text{port}} : name_{\text{library}} \ldots}$</td>
<td>memory</td>
</tr>
</tbody>
</table>

**Keyboard Access:** 🔄 LIBRARY PVARS

**Affected by Flags:** None

**Remarks:** The port number, $n_{\text{port}}$, must be in the range from 0 to 33.

- If $n_{\text{port}} = 0$, then *memory* is bytes of available main RAM.
- If the port contains independent RAM, then *memory* is bytes of available RAM in that port.
- If the port contains merged RAM, then *memory* is "SYSRAM".
- If the port contains ROM, then *memory* is "ROM".
- If the port is empty, then the message Port Not Available appears.

**Related Commands:** PVARS, VARS
**PVIEW**

**PICT View Command:** Displays PICT with the specified coordinate at the upper left corner of the graphics display.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x, y))</td>
<td>→</td>
<td>({}, #n #m)</td>
</tr>
<tr>
<td>{ }</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- PRG NXT OUT PVIEW
- PRG PICT NXT PVIEW

**Affected by Flags:** None

**Remarks:** PICT must fill the entire display on execution of PVIEW. Thus, if a position other than the upper left corner of PICT is specified, PICT must be large enough to fill a rectangle that extends 131 pixels to the right and 64 pixels down.

If PVIEW is executed from a program with a coordinate argument (versus an empty list), the graphics display persists only until the keyboard is ready for input (for example, until the end of program execution). However, the FREEZE command freezes the display until a key is pressed.

If PVIEW is executed with an empty list argument, PICT is centered in the graphics display with scrolling mode activated. In this case, the graphics display persists until CANCEL is pressed.

PVIEW does not activate the graphics cursor or the Picture menu. To activate the graphics cursor and Picture menu, execute PICTURE.

**Example:** The program

```
{ { #0d #0d } PVIEW \# FREEZE }
```

displays PICT in the graphics display with coordinates \({ #0d #0d }\) in the upper left corner of the display, then freezes the full display until a key is pressed.
PVIEVERelated Commands: FREEZE, PICTURE, TEXT

PWRFIT

Power Curve Fit Command: Stores PWRFIT as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the power curve fitting model.

Keyboard Access: (STAT) [ΣPAR] [MODL] PWRFIT

Affected by Flags: None

Remarks: LINFIT is the default specification in ΣPAR. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, EXPFIT, LINFIT, LOGFIT, LR

PX→C

Pixel to Complex Command: Converts the specified pixel coordinates to user-unit coordinates.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ #n #m }</td>
<td>→</td>
<td>(x, y)</td>
</tr>
</tbody>
</table>

Keyboard Access: PRG PICT NXT PX+C

Affected by Flags: None

Remarks: The user-unit coordinates are derived from the (xmin, ymin) and (xmax, ymax) parameters in the reserved variable PPAR. The coordinates correspond to the geometrical center of the pixel.

Related Commands: C→PX
To Quotient Command: Returns a rational form of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>→</td>
<td>'a/b'</td>
</tr>
<tr>
<td>((x,y))</td>
<td>→</td>
<td>'a/b+c/d*i'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>→</td>
<td>'symb_2'</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \) SYMBOLIC \( \) NXT → \( Q \)

Affected by Flags: Number Display (–45 to –50)

Remarks: The rational result is a “best guess”, since there might be more than one rational expression consistent with the argument. \( Q \) finds a quotient of integers that agrees with the argument to within the number of decimal places specified by the display format mode.

\( Q \) also acts on numbers that are part of algebraic expressions or equations.

Example: \('Y+2.5' \( Q \) returns \('Y+5/2'\).

Related Commands: \( Q\pi, / \)

To Quotient Times \( \pi \) Command: Returns a rational form of the argument, or a rational form of the argument with \( \pi \) factored out, whichever yields the smaller denominator.
Table:

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$→$</td>
<td>'${a/b}^\pi$'</td>
</tr>
<tr>
<td>$x$</td>
<td>$→$</td>
<td>'${a/b}$'</td>
</tr>
<tr>
<td>'${symb}_1$'</td>
<td>$→$</td>
<td>'${symb}_2$'</td>
</tr>
<tr>
<td>$(x,y)$</td>
<td>$→$</td>
<td>'${a/b}^\pi + c/d^\pi \cdot i$'</td>
</tr>
<tr>
<td>$(x,y)$</td>
<td>$→$</td>
<td>'${a/b} + c/d \cdot i$'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [Symbolic] [NXT] [π]

**Affected by Flags:** Number Format (−45 to −50)

**Remarks:** →Qπ computes two quotients (rational expressions) and compares them: the quotient of the argument, and the quotient of the argument divided by π. It returns the fraction with the smaller denominator; if the argument was divided by π, then π is a factor in the result.

The rational result is a “best guess,” since there might be more than one rational expression consistent with the argument. →Qπ finds a quotient of integers that agrees with the argument to the number of decimal places specified by the display format mode.

→Qπ also acts on numbers that are part of algebraic expressions or equations.

For a complex argument, the real or imaginary part (or both) can have π as a factor.

**Example:** In Fix mode to four decimal places, $6.2832 \rightarrow Q \pi$ returns '2\pi'. In Standard mode, however, $6.2832 \rightarrow Q \pi$ returns $3927/625$.

**Related Commands:** →Q, /, π
QR

**QR Factorization of a Matrix Command:** Returns the QR factorization of an \( n \times m \) matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>([-\rightarrow-])</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\text{matrix } J])</td>
<td>([-\rightarrow-])</td>
<td>([\text{matrix } J]_Q)</td>
<td>([\text{matrix } J]_R)</td>
<td>([\text{matrix } J]_P)</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{MTH MATR FACTR } QR\)

**Affected by Flags:** None

**Remarks:** QR factors \( m \times n \) matrix \( A \) into three matrices:
- \( Q \) is an \( m \times m \) orthogonal matrix.
- \( R \) is an \( m \times n \) upper trapezoidal matrix.
- \( P \) is a \( n \times n \) permutation matrix.

Where \( A \times P = Q \times R \).

**Related Commands:** LQ, LSQ

QUAD

**Solve Quadratic Equation Command:** Solves an algebraic object \( 'symb_1' \) for the variable \( global \), and returns an expression \( 'symb_2' \) representing the solution.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>([-\rightarrow-])</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 'symb_1' )</td>
<td>( 'global' )</td>
<td>([-\rightarrow-])</td>
<td>( 'symb_2' )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{SYM} \quad \text{QUAD}\)

**Affected by Flags:** Principal Solution \((-1)\)
**QUAD**

**Remarks:** QUAD calculates the second-degree Taylor series approximation of \( 'symb_1' \) to convert it to quadratic form. The solution \( 'symb_2' \) is exact if \( 'symb_1' \) is second degree or less in \( global \).

Since QUAD evaluates \( 'symb_1' \), any variables in \( 'symb_1' \) other than \( global \) should not exist in the current directory if they are to remain in the solution as formal variables.

QUAD generally does not work if \( global \) needs units to satisfy the equation.

**Example:** \( 'A*X^2+B*X+C=0' 'X' \) QUAD returns

\[
X = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}
\]

which reduces to the familiar quadratic solution:

\[
X = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}
\]

**Related Commands:** COLCT, EXPAN, ISOL, SHOW

---

**QUOTE**

**Quote Argument Function:** Returns its argument unevaluated.

\( {} \)

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 'symb' )</td>
<td>→</td>
<td>( 'symb' )</td>
</tr>
<tr>
<td>( obj )</td>
<td>→</td>
<td>( obj )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{SYMBOLOC} \) \( \text{NXT} \) \( \text{NXT} \) \( \text{QUOT} \)

**Affected by Flags:** None

**Remarks:** When an algebraic expression is evaluated, the arguments to a function in the expression are evaluated before the function. For example, when \( '\text{SIN}(X)' \) is evaluated, the name \( X \) is evaluated first, and the result is left on the stack as the argument for SIN.
This process creates a problem for functions that require symbolic arguments. For example, the function ∫ requires as one of its arguments a name specifying the variable of integration. If evaluating an integral expression caused the name to be evaluated, the result of evaluation would be left on the stack for ∫, rather than the name itself. To avoid this problem, the HP 48 automatically (and invisibly) quotes such arguments. When the quoted argument is evaluated, the unquoted argument is returned.

If a user-defined function takes symbolic arguments, those arguments must be quoted using QUOTE, as demonstrated in the following example.

**Example:** The following user-defined function *ArcLen* calculates the arc length of a function:

```plaintext
«
  → start end expr var
  «
    start end
    expr var ∂ SQ 1 + ∫
    var ∫
  »
»

ENTER (ArcLen STO)
```

To use this user-defined function in an algebraic expression, the symbolic arguments must be quoted:

'ArcLen(0,π,QUOTE(SIN(x)),QUOTE(x))'

**Related Commands:** APPLY, | (Where)
RAD

Radians Mode Command: Sets Radians angle mode.

Keyboard Access:

- 🔄 RAD
- 🔄 MODES ANGL RAD

Affected by Flags: None

Remarks: RAD sets flag —17 and clears flag —18, and displays the
RAD annunciator.

In Radians angle mode, real-number arguments that represent angles
are interpreted as radians, and real-number results that represent
angles are expressed in radians.

Related Commands: DEG, GRAD

RAND

Random Number Command: Returns a pseudo-random number
generated using a seed value, and updates the seed value.

| Level 1 | → | Level 1 | → | $x_{\text{random}}$ |

Keyboard Access: 🔄 MTH MTH NXT PROB RAND

Affected by Flags: None

Remarks: The HP 48 uses a linear congruential method and a seed
value to generate a random number $x_{\text{random}}$ in the range $0 \leq x < 1$.
Each succeeding execution of RAND returns a value computed from a
seed value based upon the previous RAND value. (Use RDZ to change
the seed.)

Related Commands: COMB, PERM, RDZ, !
RANK
Matrix Rank Command: Returns the rank of a rectangular matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[ matrix ]]</td>
<td>→</td>
<td>( n_{\text{rank}} )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{MTH MATR NORM NXT RANK} \)

Affected by Flags: Singular Value (–54)

Remarks: Rank is computed by calculating the singular values of the matrix and counting the number of nonnegligible values. If all computed singular values are zero, RANK returns zero. Otherwise RANK consults flag –54 as follows:

- If flag –54 is clear (the default), RANK counts all computed singular values that are less than or equal to \( 1.0 \times 10^{-14} \) times the largest computed singular value.
- If flag –54 is set, RANK counts all nonzero computed singular values.

Related Commands: LQ, LSQ, QR

RANM
Random Matrix Command: Returns a matrix of specified dimensions that contains random integers in the range \(-9\) through \(9\).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ ( m ) ( n ) }</td>
<td>→</td>
<td>( [[ \text{random matrix} ]]_{m \times n} )</td>
</tr>
<tr>
<td>( [[ \text{matrix} ]]_{m \times n} )</td>
<td>→</td>
<td>( [[ \text{random matrix} ]]_{m \times n} )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{MTH MATR MAKE RANM} \)
RANM

Affected by Flags: None

Remarks: The probability of a particular nonzero digit occurring is 0.05; the probability of 0 occurring is 0.1.

Related Commands: RAND, RDZ

RATIO

Prefix Divide Function: Prefix form of / (divide) generated by the EquationWriter application.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>$z_2$</td>
<td>→</td>
<td>$z_1 / z_2$</td>
</tr>
<tr>
<td>[ array ]</td>
<td>[ matrix ]</td>
<td>→</td>
<td>[ array x matrix$^{-1}$ ]</td>
</tr>
<tr>
<td>$z$</td>
<td>'symb'</td>
<td>→</td>
<td>'z/symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$z$</td>
<td>→</td>
<td>'symb/z'</td>
</tr>
<tr>
<td>'symb$^1$'</td>
<td>'symb$^2$'</td>
<td>→</td>
<td>'symb$^1 / symb^2$'</td>
</tr>
<tr>
<td>#n$^1$</td>
<td>n$^2$</td>
<td>→</td>
<td>#n$^3$</td>
</tr>
<tr>
<td>n$^1$</td>
<td>#n$^2$</td>
<td>→</td>
<td>#n$^3$</td>
</tr>
<tr>
<td>#n$^1$</td>
<td>#n$^2$</td>
<td>→</td>
<td>#n$^3$</td>
</tr>
<tr>
<td>x$^1$.unit$^1$</td>
<td>y$^1$.unit$^2$</td>
<td>→</td>
<td>(x/y).unit$^1$ /unit$^2$</td>
</tr>
<tr>
<td>x</td>
<td>y$^1$.unit</td>
<td>→</td>
<td>(x/y).unit$^1$</td>
</tr>
<tr>
<td>x$^1$.unit</td>
<td>y$^2$.unit</td>
<td>→</td>
<td>(x/y).unit$^2$</td>
</tr>
<tr>
<td>'symb'</td>
<td>x$^1$.unit</td>
<td>→</td>
<td>'symb/x$^1$.unit'</td>
</tr>
<tr>
<td>x$^1$.unit</td>
<td>'symb'</td>
<td>→</td>
<td>'x$^1$.unit/symb'</td>
</tr>
</tbody>
</table>

Keyboard Access: None. Must be typed in.

Affected by Flags: None
**Remarks:** RATIO is identical to / (divide), except that, in algebraic syntax, RATIO is a *prefix* function, while / is an *infix* function. For example, \('\text{RATIO}(x,2)'\) is equivalent to \('x/2'\).

RATIO is generated internally by the EquationWriter application when \(\text{[\text{\textbullet\text{\textbullet\textbullet}}}\) is used to start a numerator. It provides no additional functionality to / and appears externally only in the string that the EquationWriter application leaves on the stack when \(\text{[\text{\textbullet\text{\textbullet\textbullet}}}\) is pressed or when the calculator runs out of memory.

**Related Commands:** /

---

**RCEQ**

**Recall from EQ Command:** Returns the unevaluated contents of the reserved variable \(EQ\) from the current directory.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>(\rightarrow)</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\rightarrow)</td>
<td>(obj_{EQ})</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- \(<\text{PLOT}\>\quad EQ\)
- \(<\text{PLOT}\>\text{NXT}\>\quad 3D\quad EQ\)
- \(<\text{PLOT}\>\quad EQ\quad \text{(program-entry mode)}\)
- \(<\text{PLOT}\>\text{NXT}\>\quad 3D\quad \rightarrow\quad EQ\quad \text{(program-entry mode)}\)

**Affected by Flags:** None

**Remarks:** To recall the contents of \(EQ\) from a parent directory (when \(EQ\) doesn’t exist in the current directory) evaluate the name \(EQ\).

**Related Commands:** STEQ

---
**RCI**

**Multiply Row by Constant Command:** Multiplies row \( n \) of a matrix (or element \( n \) of a vector) by a constant \( x_{\text{factor}} \), and returns the modified matrix.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([ \text{matrix} ]_1)</td>
<td>(x_{\text{factor}})</td>
<td>(n_{\text{row number}})</td>
<td>→</td>
<td>([ \text{matrix} ]_2)</td>
</tr>
<tr>
<td>([ \text{vector} ]_1)</td>
<td>(x_{\text{factor}})</td>
<td>(n_{\text{element number}})</td>
<td>→</td>
<td>([ \text{vector} ]_2)</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{MTH MATR ROW RCI}\)

**Affected by Flags:** None

**Remarks:** RCI rounds the row number to the nearest integer, and treats vector arguments as column vectors.

**Related Commands:** RCIJ

**RCIJ**

**Add Multiplied Row Command:** Multiplies row \( i \) of a matrix by a constant \( x_{\text{factor}} \), adds this product to row \( j \) of the matrix, and returns the modified matrix. Or, multiplies element \( i \) of a vector by a constant \( x_{\text{factor}} \), adds this product to element \( j \) of the vector, and returns the modified vector.

<table>
<thead>
<tr>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([ \text{matrix} ]_1)</td>
<td>(x_{\text{factor}})</td>
<td>(n_{\text{row i}})</td>
<td>(n_{\text{row j}})</td>
<td>→</td>
<td>([ \text{matrix} ]_2)</td>
</tr>
<tr>
<td>([ \text{vector} ]_1)</td>
<td>(x_{\text{factor}})</td>
<td>(n_{\text{element i}})</td>
<td>(n_{\text{element j}})</td>
<td>→</td>
<td>([ \text{vector} ]_2)</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{MTH MATR ROW RCIJ}\)

3-260 Command Reference
**Affected by Flags:** None

**Remarks:** RCIJ rounds the row numbers to the nearest integer, and treats vector arguments as column vectors.

**Related Commands:** RCI

---

**RCL**

**Recall Command:** Returns the unevaluated contents of a specified variable or plug-in object.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td>obj</td>
</tr>
<tr>
<td>PICT</td>
<td>→</td>
<td>grob</td>
</tr>
<tr>
<td>:n_port :n_library</td>
<td>→</td>
<td>obj</td>
</tr>
<tr>
<td>:n_port :name_backup</td>
<td>→</td>
<td>obj</td>
</tr>
</tbody>
</table>

**Keyboard Access:** ()(RCL)

**Affected by Flags:** None

**Remarks:** RCL searches the entire current path, starting with the current directory. To search a different path, specify `{ path name }`, where path is the new path to the variable name. The path subdirectory does not become the current subdirectory (unlike EVAL).

To recall a library or backup object, tag the library number or backup name with the appropriate port number (n_port), which must be an integer in the range 0 to 33. (A library can be recalled from RAM only.) Recalling a backup object brings a copy of its contents to the stack, not the entire backup object.

To search for a backup object, replace the port number with the wildcard character *, in which case the HP 48 will search (in order) ports 33 through 0, and the main memory for the named backup object.

**Related Commands:** STO
RCLALARM

Recall Alarm Command: Recalls a specified alarm.

\[
\begin{array}{|c|c|}
\hline
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
n_{\text{index}} & \rightarrow & \{ \text{date time } obj_{\text{action}} \ x_{\text{repeat}} \} \\
\hline
\end{array}
\]

Keyboard Access: \text{\uline{TIME} ALRM RCLAL}

Affected by Flags: None

Remarks: \( obj_{\text{action}} \) is the alarm execution action. If an execution action was not specified, \( obj_{\text{action}} \) defaults to an empty string.

\( x_{\text{repeat}} \) is the repeat interval in clock ticks, where 1 clock tick equals \( 1/8192 \) second. If a repeat interval was not specified, the default is 0.

Related Commands: \text{DELALARM, FINDALARM, STOALARM}

RCLF

Recall Flags Command: Returns a list containing two 64-bit binary integers representing the states of the 64 system and user flags, respectively.

\[
\begin{array}{|c|c|}
\hline
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
& \rightarrow & \{ \#n_{\text{system}} \ #n_{\text{user}} \} \\
\hline
\end{array}
\]

Keyboard Access: \text{\uline{MODES} FLAG\uline{NXT} RCLF}

Affected by Flags: Binary Integer Wordsize \((-5 \text{ through } -10)\)

The current wordsize must be 64 bits (the default wordsize) to recall the states of all 64 user flags and 64 system flags. If the current wordsize is 32, for example, RCLF returns two 32-bit binary integers.
**Remarks:** A bit with value 1 indicates that the corresponding flag is set; a bit with value 0 indicates that the corresponding flag is clear. The rightmost (least significant) bit of \$n_{\text{system}} \text{ and } \#n_{\text{user}}\text{ indicate the states of system flag } -1 \text{ and user flag } +1, \text{ respectively.}

Used with STOF, RCLF lets a program that alters the state of a flag or flags during program execution preserve the pre-program-execution flag status.

**Related Commands:** STOF

---

**RCLKEYS**

**Recall Key Assignments Command:** Returns the current user key assignments. This includes an % if the standard definitions are active (not suppressed) for those keys without user key assignments.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow )</td>
<td>( { \text{obj}_1 \ \text{xkey}_1 \ \ldots \ \text{obj}_n \ \text{xkey}_n } )</td>
<td>( \rightarrow )</td>
</tr>
<tr>
<td>( \rightarrow )</td>
<td>( { \text{S obj}_1 \ \text{xkey}_1 \ \ldots \ \text{obj}_n \ \text{xkey}_n } )</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MODES KEYS RCLK} \)

**Affected by Flags:** User-Mode Lock (–61) and User Mode (–62) affect the status of the user keyboard.

**Remarks:** The argument \( x_{\text{key}} \) is a real number of the form \( rc.p \) specifying the key by its row number \( r \), its column number \( c \), and its plane (shift) \( p \). (For a definition of plane, see the entry for ASN.)

**Related Commands:** ASN, DELKEYS, STOKEYS
RCLMENU

Recall Menu Number Command: Returns the menu number of the currently displayed menu.

Keyboard Access: \texttt{\textasciitilde \textasciitilde MODES MENU RCLM}

Affected by Flags: None

Remarks: $x_{\text{menu}}$ has the form $mm.pp$, where $mm$ is the menu number and $pp$ is the page of the menu. See the MENU entry for a list of the HP 48 built-in menus and the corresponding menu numbers.

Executing RCLMENU when the current menu is a user-defined menu (built by TMENU) returns \texttt{\textasciitilde \textasciitilde 1} (in 2 Fix mode), indicating "Last menu".

Example: If the third page of the PRG DSPL menu is currently active, RCLMENU returns \texttt{13. \textasciitilde \textasciitilde 3} (in 2 Fix mode).

Related Commands: MENU, TMENU

RCL∑

Recall Sigma Command: Returns the current statistics matrix (the contents of reserved variable $\Sigma DAT$) from the current directory.

Keyboard Access:

\texttt{\textasciitilde \textasciitilde PLOT NXT STAT ΣDAT}
Recall Wordsize Command: Returns the current wordsize in bits (1 through 64).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>n</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{MTH} \text{ BASE} \text{ NXT} \text{ RCWS} \)

Affected by Flags: Binary Integer Wordsize \((-5\) through \(-10\)), Binary Integer Base \((-11, -12)\)

Related Commands: \( \text{BIN, DEC, HEX, OCT, STWS} \)
RDM

Redimension Array Command: Rearranges the elements of the argument according to specified dimensions.

Keyboard Access: [MTH] [MATR] [MAKE] [RDM]

Affected by Flags: None

Remarks: If the list contains a single number \( n_{\text{elements}} \), the result is an \( n \)-element vector. If the list contains two numbers \( n_{\text{rows}} \) and \( m_{\text{cols}} \), the result is an \( n \times m \) matrix.

Elements taken from the argument vector or matrix preserve the same row order in the resulting vector or matrix. If the result is dimensioned to contain fewer elements than the argument vector or matrix, excess elements from the argument vector or matrix at the end of the row order are discarded. If the result is dimensioned to contain more elements than the argument vector or matrix, the additional elements in the result at the end of the row order are filled with zeros (if the argument is complex).

If the argument vector or matrix is specified by \texttt{global}, the result replaces the argument as the contents of the variable.

Examples: \( [2 4 6 8] (2 2) \) RDM returns \( [[2 4][6 8]] \).
\( [[2 3 4][1 6 9]]8 \) RDM returns \( [2 3 4 1 6 9 0 0] \).

Related Commands: TRN
RDZ

Randomize Command: Uses a real number $x_{seed}$ as a seed for the RAND command.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{seed}$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: [MTH] [NXT] PROB [RDZ]

Affected by Flags: None

Remarks: If the argument is $\emptyset$, a random value based on the system clock is used as the seed.

Related Commands: COMB, PERM, RAND, !

RE

Real Part Function: Returns the real part of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>→</td>
<td>$x$</td>
</tr>
<tr>
<td>$x_{\text{unit}}$</td>
<td>→</td>
<td>$x$</td>
</tr>
<tr>
<td>$(x, y)$</td>
<td>→</td>
<td>$x$</td>
</tr>
<tr>
<td>[ R-array ]</td>
<td>→</td>
<td>[ R-array ]</td>
</tr>
<tr>
<td>[ C-array ]</td>
<td>→</td>
<td>[ R-array ]</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'RE(symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access: [MTH] [NXT] COMPL [RE]

Affected by Flags: Numerical Results (−3)
**RE**

**Remarks:** If the argument is a vector or matrix, RE returns a real array, the elements of which are equal to the real parts of the corresponding elements of the argument array.

**Related Commands:** C→R, IM, R→C

---

**RECN**

**Receive Renamed Object Command:** Prepares the HP 48 to receive a file from another Kermit device, and to store the file in a specified variable.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
</table>
| 'name'        | → |"
| "name"        | → |

**Keyboard Access:** [I/O] [NXT] **RECN**

**Affected by Flags:** I/O Device (−33), I/O Data Format (−35), RECV Overwrite (−36), I/O Messages (−39)

When an HP 48 sends an object, it automatically appends a header that tells a receiving HP 48 whether to use ASCII or binary mode. Flag −35 has an effect only if this header is not present.

**Remarks:** RECN is identical to RECV except that the name under which the received data is stored is specified in the stack.

**Related Commands:** BAUD, CKSM, CLOSEIO, FINISH, KERRM, KGET, PARITY, RECV, SEND, SERVER, TRANSIO
RECT

Rectangular Mode Command:  Sets Rectangular coordinate mode.

Keyboard Access:

- MTH VECTR NXT RECT
- Modes ANGL RECT

Affected by Flags:  None

Remarks:  RECT clears flags —15 and —16, and clears the θ and θ annunciators.

In Rectangular mode, vectors are displayed as rectangular components. Therefore, a 3D vector would appear as \([X Y Z]\).

Related Commands: CYLIN, SPHERE

RECV

Receive Object Command:  Instructs the HP 48 to look for a named file from another Kermit device. The received file is stored in a variable named by the sender.

Keyboard Access:  I/O REC

Affected by Flags:  I/O Device (—33), I/O Data Format (—35), RECV Overwrite (—36), I/O Messages (—39)

When an HP 48 sends an object, it automatically appends a header that tells a receiving HP 48 whether to use ASCII or binary mode. Flag —35 has an effect only if this header is not present.

Remarks:  Since the HP 48 does not normally look for incoming Kermit files, you must use RECV to tell it to do so.

Related Commands:  BAUD, CKSM, FINISH, KGET, PARITY, RECN, SEND, SERVER, TRANSIO
**REPEAT**

**REPEAT Command:** Starts loop clause in WHILE ... REPEAT ... END indefinite loop structure.

See the WHILE entry for syntax information.

**Keyboard Access:** PRG BRCH WHILE REPE

**Remarks:** See the WHILE entry for more information.

**Related Commands:** END, WHILE

---

**REPL**

**Replace Command:** Replaces a portion of the level 3 target object with the level 1 object, beginning at a position specified in level 2.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[ matrix ]]_1</td>
<td>_n_position</td>
<td>[[ matrix ]]_2</td>
<td>→</td>
<td>[[ matrix ]]_3</td>
</tr>
<tr>
<td>[[ matrix ]]_1</td>
<td>{ _n_row _n_column }</td>
<td>[[ matrix ]]_2</td>
<td>→</td>
<td>[[ matrix ]]_3</td>
</tr>
<tr>
<td>[ vector ]_1</td>
<td>_n_position</td>
<td>[ vector ]_2</td>
<td>→</td>
<td>[ vector ]_3</td>
</tr>
<tr>
<td>{ list_target }</td>
<td>_n_position</td>
<td>{ list_1 }</td>
<td>→</td>
<td>{ list_result }</td>
</tr>
<tr>
<td>&quot;string_target&quot;</td>
<td>_n_position</td>
<td>&quot;string_1&quot;</td>
<td>→</td>
<td>&quot;string_result&quot;</td>
</tr>
<tr>
<td>grob_target</td>
<td>#n #m</td>
<td>grob_1</td>
<td>→</td>
<td>grob_result</td>
</tr>
<tr>
<td>grob_target</td>
<td>(x,y)</td>
<td>grob_1</td>
<td>→</td>
<td>grob_result</td>
</tr>
<tr>
<td>PICT</td>
<td>#n #m</td>
<td></td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>PICT</td>
<td>(x,y)</td>
<td></td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- [CHARS] REPL
- PRG LIST REPL
- PRG GROB REPL
- MTH MATR MAKE NXT REPL

3-270 Command Reference
Affected by Flags: None

Remarks: 
For arrays, \( n_{\text{position}} \) counts in row order. For matrices, \( n_{\text{position}} \) specifies the new location of the upper left-hand element of the replacement matrix.

For graphics objects, the upper left corner of \( \text{grob}_1 \) is positioned at the user-unit or pixel coordinates \( (x, y) \) or \( (\#n \#m) \). From there, it overwrites a rectangular portion of \( \text{grob}_{\text{target}} \) or \( \text{PICT} \). If \( \text{grob}_1 \) extends past \( \text{grob}_{\text{target}} \) or \( \text{PICT} \) in either direction, it is truncated in that direction. If the specified coordinate is not on the target graphics object, the target graphics object does not change.

Examples:
\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{bmatrix}
\]
6 \( \begin{bmatrix}
2 & 2 \\
2 & 2 \\
\end{bmatrix} \) REPL returns
\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 2 & 2 & 1 \\
1 & 2 & 2 & 1 \\
\end{bmatrix}. \\
\]

\( \text{"ABCDE"} \) 2 \( \text{"FG"} \) REPL returns \( \text{"ABCDGF"} \).

ERASE \( \text{PICT}(0, 0) \#5d \#5d \) BLANK NEG REPL replaces a portion of \( \text{PICT} \) with a 5 \( \times \) 5 graphics object, each of whose pixels is on (dark), and whose upper left corner is positioned at (0, 0) in \( \text{PICT} \).

Related Commands: CHR, GOR, GXOR, NUM, POS, SIZE, SUB

RES

Resolution Command: 
Specifies the resolution of mathematical and statistical plots, where the resolution is the interval between values of the independent variable used to generate the plot.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{\text{interval}} )</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>#( n_{\text{interval}} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{←} \text{PLT} \) PPAR RES
RES

Affected by Flags: None

Remarks: A real number \( n_{\text{interval}} \) specifies the interval in user units. A binary integer \( #n_{\text{interval}} \) specifies the interval in pixels.

The resolution is stored as the fourth item in PPAR, with default value 0. The interpretation of the default value is summarized in the following table.

<table>
<thead>
<tr>
<th>Plot Type</th>
<th>Default Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAR</td>
<td>10 pixels (bar width = 10 pixel columns)</td>
</tr>
<tr>
<td>DIFFEQ</td>
<td>unlimited: step size is not constrained</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>1 pixel (plots a point in every column of pixels)</td>
</tr>
<tr>
<td>CONIC</td>
<td>1 pixel (plots a point in every column of pixels)</td>
</tr>
<tr>
<td>TRUTH</td>
<td>1 pixel (plots a point in every column of pixels)</td>
</tr>
<tr>
<td>GRIDMAP</td>
<td>RES does not apply</td>
</tr>
<tr>
<td>HISTOGRAM</td>
<td>10 pixels (bin width = 10 pixel columns)</td>
</tr>
<tr>
<td>PARAMETRIC</td>
<td>[independent variable range in user units]/130</td>
</tr>
<tr>
<td>PARSURFACE</td>
<td>RES does not apply</td>
</tr>
<tr>
<td>PCONTOUR</td>
<td>RES does not apply</td>
</tr>
<tr>
<td>POLAR</td>
<td>2°, 2 grads, or ( \pi/90 ) radians</td>
</tr>
<tr>
<td>SCATTER</td>
<td>RES does not apply</td>
</tr>
<tr>
<td>SLOPEFIELD</td>
<td>RES does not apply</td>
</tr>
<tr>
<td>WIREFRAME</td>
<td>RES does not apply</td>
</tr>
<tr>
<td>YSLICE</td>
<td>1 pixel (plots a point in every column of pixels)</td>
</tr>
</tbody>
</table>

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE
RESTORE

Restore HOME Command: Replaces the current HOME directory with the specified backup copy 
\( :n_{port} :name_{backup} \) previously created by ARCHIVE.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>( :n_{port} :name_{backup} )</th>
<th>( backup )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>( \rightarrow )</td>
<td>( \rightarrow )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \langle \rangle \) MEMORY NXT RESTORE

Affected by Flags: None

Remarks: The specified port number must be in the range 0 to 33. Ports 1 and 2 must be configured as independent RAM (see FREE).

To restore a HOME directory that was saved on a remote system using \( :IO:name \) ARCHIVE, put the backup object itself on the stack and execute RESTORE.

Example: To restore a HOME directory that was saved as the file AUG1 on a remote system, execute 'AUG1' SEND on the remote system, then execute the following on the HP 48:

\[ \text{RECY 'AUG1' RCL RESTORE} \]

Related Commands: ARCHIVE
**REVLIST**

**Reverse List Command:** Reverses the order of the elements in a list.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ \textit{obj}_n \ldots \textit{obj}_1 }</td>
<td>→</td>
<td>{ \textit{obj}_1 \ldots \textit{obj}_n }</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- PRG \texttt{LIST} \texttt{PROC} \texttt{REVLI}
- MTH \texttt{LIST} \texttt{REVLI}

**Affected by Flags:** None

**Related Commands:** ↑\texttt{SORT}

---

**RKF**

**Solve for Initial Values (Runge-Kutta-Fehlberg) Command:**
Computes the solution to an initial value problem for a differential equation, using the Runge-Kutta-Fehlberg (4,5) method.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ \textit{list} }</td>
<td>\textit{x}_{\text{tol}}</td>
<td>\textit{x}_{\text{T_final}}</td>
<td>→</td>
<td>{ \textit{list} }</td>
<td>\textit{x}_{\text{tol}}</td>
</tr>
<tr>
<td>{ \textit{list} }</td>
<td>{ \textit{x}<em>{\text{tol}} \textit{x}</em>{\text{hstep}} }</td>
<td>\textit{x}_{\text{T_final}}</td>
<td>→</td>
<td>{ \textit{list} }</td>
<td>\textit{x}_{\text{tol}}</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\texttt{SOLVE} \texttt{DIFF \_ FEH \_ RUNG \_ KUTTA \_ F})

**Affected by Flags:** None

**Remarks:** RKF solves \(y'(t)=f(t,y)\), where \(y(t_0)=y_0\). The arguments and results are as follows:
\{ list \} contains three items in this order: the independent \( (t) \) and solution \( (y) \) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).

- \( x_{\text{tol}} \) sets the absolute error tolerance. If a list is used, the first value is the absolute error tolerance and the second value is the initial candidate step size.

- \( x_{\text{final}} \) specifies the final value of the independent variable.

RKF repeatedly calls RKFSTEP as it steps from the initial value to \( x_{\text{final}} \).

**Example:** Solve the following initial value problem for \( y(8) \), given that \( y(0) = 0 \):

\[
y' = \frac{1}{(1 + t^2)} - 2y^2 = f(t, y)
\]

1. Store the independent variable's initial value, 0, in \( T \).
2. Store the dependent variable's initial value, 0, in \( Y \).
3. Store the expression, \( \frac{1}{(1+t^2)} - 2y^2 \), in \( F \).
4. Enter a list containing these three items: \( \{ T \ Y \ F \} \).
5. Enter the tolerance. Use estimated decimal place accuracy as a guideline for choosing a tolerance: 0.00001.
6. Enter the final value for the independent variable: 8.

The stack should look like this:

\[
\{ T \ Y \ F \} \\
.00001 \\
8
\]

7. Press \( \text{RKF} \). (The calculation takes a moment.) The variable \( T \) now contains 8, and \( Y \) now contains the value .123077277659.

The actual answer is .123076923077, so the calculated answer has an error of approximately .00000035, well within the specified tolerance.

**Related Commands:** RKFERR, RKFSTEP, RRK, RRKSTEP, RSBERR
**RKFERR**

Error Estimate for Runge-Kutta-Fehlberg Method Command:
Returns the absolute error estimate for a given step $h$ when solving an initial value problem for a differential equation.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ list }</td>
<td>$h$</td>
<td>→</td>
<td>{ list }</td>
<td>$h$</td>
<td>$y_{\delta}$</td>
<td>error</td>
</tr>
</tbody>
</table>

**Keyboard Access:** («)(SOLVE) DIFFE RKFERR

**Affected by Flags:** None

**Remarks:** The arguments and results are as follows:
- The arguments $m$ contains three items in this order: the independent $(t)$ and solution $(y)$ variables, and the right-hand side of the differential equation (or a variable where the expression is stored).
- $h$ is a real number that specifies the step.
- $y_{\delta}$ displays the change in solution for the specified step.
- error displays the absolute error for that step. A zero error indicates that the Runge-Kutta-Fehlberg method failed and that Euler's method was used instead.

The absolute error is the absolute value of the estimated error for a scalar problem, and the row (infinity) norm of the estimated error vector for a vector problem. (The latter is a bound on the maximum error of any component of the solution.)

**Related Commands:** RKF, RKFSTEP, RRK, RRKSTEP, RSBERR
**RKFSTEP**

**Next Solution Step for RKF Command:** Computes the next solution step \( (h_{\text{next}}) \) to an initial value problem for a differential equation.

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Level 3} & \text{Level 2} & \text{Level 1} & \rightarrow & \text{Level 3} & \text{Level 2} & \text{Level 1} \\
\{ \text{list} \} & x_{\text{tol}} & h & \rightarrow & \{ \text{list} \} & x_{\text{tol}} & h_{\text{next}} \\
\end{array}
\]

**Keyboard Access:** \( \text{SOLVE} \text{ DIFFE} \text{ RKFST} \)

**Affected by Flags:** None

**Remarks:** The arguments and results are as follows:

- \{ \text{list} \} contains three items in this order: the independent \((t)\) and solution \((y)\) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).

- \(x_{\text{tol}}\) sets the tolerance value.

- \(h\) specifies the initial candidate step.

- \(h_{\text{next}}\) is the next candidate step.

The independent and solution variables must have values stored in them. RKFSTEP steps these variables to the next point upon completion.

Note that the actual step used by RKFSTEP will be less than the input value \(h\) if the global error tolerance is not satisfied by that value. If a stringent global error tolerance forces RKFSTEP to reduce its stepsize to the point that the Runge-Kutta-Fehlberg method fails, then RKFSTEP will use the Euler method to compute the next solution step and will consider the error tolerance satisfied. The Runge-Kutta-Fehlberg method will fail if the current independent variable is zero and the stepsize \(\leq 1.3 \times 10^{-498}\) or if the variable is nonzero and the stepsize is \(1.3 \times 10^{-10}\) times its magnitude.

**Related Commands:** RKF, RKFERR, RRK, RRKSTEP, RSBERR
**RL**

**Rotate Left Command:** Rotates a binary integer one bit to the left.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH BASE NXT BIT RL

**Affected by Flags:** Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

**Remarks:** The leftmost bit of \#n₁ becomes the rightmost bit of \#n₂. Related Commands: RLB, RR, RRB

**RLB**

**Rotate Left Byte Command:** Rotates a binary integer one byte to the left.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH BASE NXT BYTE RL

**Affected by Flags:** Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

**Remarks:** The leftmost byte of \#n₁ becomes the rightmost byte of \#n₂. RLB is equivalent to executing RL eight times. Related Commands: RL, RR, RRB
**RND**

**Round Function:** Rounds an object to a specified number of decimal places or significant digits, or to fit the current display format.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>$n_{\text{round}}$</td>
<td>→</td>
<td>$z_2$</td>
</tr>
<tr>
<td>$z$</td>
<td>'symb$_{\text{round}}$'</td>
<td>→</td>
<td>'RND(z,symb$_{\text{round}}$)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$n_{\text{round}}$</td>
<td>→</td>
<td>'RND(symb,n_{\text{round}})'</td>
</tr>
<tr>
<td>'symb$_1$'</td>
<td>'symb$_{\text{round}}$'</td>
<td>→</td>
<td>'RND(symb$<em>1$,symb$</em>{\text{round}}$)'</td>
</tr>
<tr>
<td>[array$_1$]</td>
<td>$n_{\text{round}}$</td>
<td>→</td>
<td>[array$_2$]</td>
</tr>
<tr>
<td>x$_{\text{unit}}$</td>
<td>$n_{\text{round}}$</td>
<td>→</td>
<td>y$_{\text{unit}}$</td>
</tr>
<tr>
<td>x$_{\text{unit}}$</td>
<td>'symb$_{\text{round}}$'</td>
<td>→</td>
<td>'RND(x_{\text{unit}},symb$_{\text{round}}$)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:**  
[MTH] [REAL] [NXT] [NXT] [RND]

**Affected by Flags:** Numerical Results (−3)

**Remarks:** $n_{\text{round}}$ (or symb$_{\text{round}}$ if flag −3 is set) controls how the level 2 argument is rounded, as follows:

<table>
<thead>
<tr>
<th>$n_{\text{round}}$ or symb$_{\text{round}}$</th>
<th>Effect on Level 2 Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 through 11</td>
<td>Rounded to $n$ decimal places.</td>
</tr>
<tr>
<td>−1 through −11</td>
<td>Rounded to $n$ significant digits.</td>
</tr>
<tr>
<td>12</td>
<td>Rounded to the current display format.</td>
</tr>
</tbody>
</table>

For complex numbers and arrays, each real number element is rounded. For unit objects, the numerical part of the object is rounded.

**Examples:**  
(4.5792, 8.1275) 2 RND returns (4.58, 8.13).  
[2.34907 3.96351 2.73453] −2 RND returns [2.3 4 2.7].

**Related Commands:** TRNC
**RNRM**

**Row Norm Command:** Returns the row norm (infinity norm) of its argument array.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
</table>
| [ array ] | → | \( x_{\text{row norm}} \)

**Keyboard Access:** \( \text{(MTH) MATR NORM RNRM} \)

**Affected by Flags:** None

**Remarks:** The row norm is the maximum (over all rows) of the sums of the absolute values of all elements in each row. For a vector, the row norm is the largest absolute value of any of its elements.

**Related Commands:** CNRM, CROSS, DET, DOT

---

**ROLL**

**Roll Objects Command:** Moves the contents of a specified level to level 1, and rolls upwards the portion of the stack beneath the specified level.

<table>
<thead>
<tr>
<th>Level ( n+1 ) ... Level 2 Level 1</th>
<th>→</th>
<th>Level ( n ) ... Level 2 Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( obj_n ) ... ( obj_1 ) ( n )</td>
<td>→</td>
<td>( obj_{n-1} ) ... ( obj_1 ) ( obj_n )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{(a)(STACK) ROLL} \)

**Affected by Flags:** None

**Remarks:** 3 \( \text{ROLL} \) is equivalent to \( \text{ROT} \).

**Related Commands:** OVER, PICK, ROLLD, ROT, SWAP
**ROLLD**

**Roll Down Command:** Moves the contents of level 1 to a specified level, and rolls downwards the portion of the stack beneath the specified level.

<table>
<thead>
<tr>
<th>Lvl n+1 ... Lvl 2</th>
<th>Lvl 1</th>
<th>Lvl n</th>
<th>Lvl n-1 ... Lvl 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj_n ... obj_1</td>
<td>n</td>
<td>obj_1</td>
<td>obj_n ... obj_2</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{\(\rightarrow\)} \text{STACK} \text{ROLLD}\)

**Affected by Flags:** None

**Related Commands:** OVER, PICK, ROLL, ROT, SWAP

---

**ROOT**

**Root-Finder Command:** Returns a real number \(x_{\text{root}}\) that is a value of the specified variable \(global\) for which the specified program or algebraic object most nearly evaluates to zero or a local extremum.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>(\rightarrow)</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\ll \text{program} \gg)</td>
<td>'global'</td>
<td>guess</td>
<td>(\rightarrow)</td>
<td>(x_{\text{root}})</td>
</tr>
<tr>
<td>(\ll \text{program} \gg)</td>
<td>'global'</td>
<td>{guesses}</td>
<td>(\rightarrow)</td>
<td>(x_{\text{root}})</td>
</tr>
<tr>
<td>'symb'</td>
<td>'global'</td>
<td>guess</td>
<td>(\rightarrow)</td>
<td>(x_{\text{root}})</td>
</tr>
<tr>
<td>'symb'</td>
<td>'global'</td>
<td>{guesses}</td>
<td>(\rightarrow)</td>
<td>(x_{\text{root}})</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{\(\leftarrow\)} \text{SOLVE} \text{ROOT} \text{ROOT}\)

**Affected by Flags:** None

**Remarks:** ROOT is the programmable form of the HP Solve application.
**ROOT**

*guess* is an initial estimate of the solution. ROOT produces an error if it cannot find a solution, returning the message *Bad Guess(es)* if one or more of the guesses lie outside the domain of the equation, or returns the message *Constant ?* if the equation returns the same value at every sample point. ROOT does *not* return interpretive messages when a root is found.

---

**ROT**

**Rotate Objects Command:** Rotates the first three objects on the stack, moving the object in level 3 to level 1.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj₃</td>
<td>obj₂</td>
<td>obj₁</td>
<td>→</td>
<td>obj₂</td>
<td>obj₁</td>
<td>obj₃</td>
</tr>
</tbody>
</table>

**Keyboard Access:** ⬅️STACK ROT

**Affected by Flags:** None

**Remarks:** ROT is equivalent to 3 ROLL.

**Related Commands:** OVER, PICK, ROLL, ROLLD, SWAP

---

**→ROW**

**Matrix to Rows Command:** Transforms a matrix into a series of row vectors and returns the vectors and a row count, or transforms a vector into its elements and returns the elements and an element count.
**ROW+**

**Insert Row Command:** Inserts an array into a matrix (or one or more numbers into a vector) at the position indicated by \( n_{\text{index}} \), and returns the modified matrix (or vector).

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([ [\text{matrix}]_1 )</td>
<td>([ \text{matrix} ]_2 )</td>
<td>( n_{\text{index}} )</td>
<td>( n_{\text{index}} )</td>
</tr>
<tr>
<td>([ [\text{matrix}]_1 )</td>
<td>([ \text{vector} ]_{\text{row}} )</td>
<td>( n_{\text{index}} )</td>
<td>( n_{\text{index}} )</td>
</tr>
<tr>
<td>([ \text{vector} ]_1 )</td>
<td>( n_{\text{element}} )</td>
<td>( n_{\text{index}} )</td>
<td>( [ \text{vector} ]_2 )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (MTH) MATR ROW ROW+

**Affected by Flags:** None

**Related Commands:** COL-, COL+, ROW-, RSWP

**Remarks:** The inserted array must have the same number of columns as the target array.

\( n_{\text{index}} \) is rounded to the nearest integer. The original array is redimensioned to include the new columns or elements, and the elements at and below the insertion point are shifted down.
Delete Row Command: Deletes row \( n \) of a matrix (or element \( n \) of a vector), and returns the modified matrix (or vector) and the deleted row (or element).

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([ [ \text{matrix} ]_1 )]</td>
<td>( n_{\text{row}} )</td>
<td>→</td>
<td>([ [ \text{matrix} ]_2 )]</td>
<td>([ \text{vector} ]_{\text{row}} )</td>
</tr>
<tr>
<td>[ ( \text{vector} )_1 ]</td>
<td>( n_{\text{element}} )</td>
<td>→</td>
<td>[ ( \text{vector} )_2 ]</td>
<td>element( n )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{\texttt{MTH MATR ROW ROW}} \)

Affected by Flags: None

Remarks: \( n_{\text{row}} \) or \( n_{\text{element}} \) is rounded to the nearest integer.

Related Commands: \( \text{COL, COL+, ROW+, RSWP} \)

Rows to Matrix Command: Transforms a series of row vectors and a row count into a matrix containing those rows, or transforms a sequence of numbers and an element count into a vector with those numbers as elements.

<table>
<thead>
<tr>
<th>Level( n_{+1} \ldots )</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([ \text{vector} ]_{\text{row}1} )]</td>
<td>( n_{\text{rowcount}} )</td>
<td>→</td>
<td>([ [ \text{matrix} ] )]</td>
<td></td>
</tr>
<tr>
<td>( \text{element}_{1} )</td>
<td>( \text{element}_{n} )</td>
<td>( n_{\text{elementcount}} )</td>
<td>→</td>
<td>([ \text{vector} ]_{\text{column}} )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{\texttt{MTH MATR ROW ROW}} \)

Affected by Flags: None

Related Commands: \( \text{→COL, COL→, →ROW} \)
RR

Rotate Right Command: Rotates a binary integer one bit to the right.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
</tbody>
</table>

Keyboard Access: MTH BASE NXT BIT RRB

Affected by Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12).

Remarks: The rightmost bit of #n₁ becomes the leftmost bit of #n₂.

Related Commands: RL, RLB, RRB

RRB

Rotate Right Byte Command: Rotates a binary integer one byte to the right.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
</tbody>
</table>

Keyboard Access: MTH BASE NXT BYT RRB

Affected by Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12).

Remarks: The rightmost byte of #n₁ becomes the leftmost byte of #n₂. RRB is equivalent to doing RR eight times.

Related Commands: RL, RLB, RR
**RREF**

**Reduced Row Echelon Form Command:** Converts a rectangular matrix to reduced row echelon form.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[ matrix ]]_A</td>
<td>→</td>
<td>[[ matrix ]]_R</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \([\text{MTH}] \text{MATR FACTR RREF} \)

**Affected by Flags:** Singular Values (—54)

**Remarks:** Converts a given matrix into reduced row echelon form. Since row echelon form is primarily used for studying systems of linear equations, RREF ignores very small pivots if system flag —54 is clear.

**Related Commands:** LU

---

**RRK**

**Solve for Initial Values (Rosenbrock, Runge-Kutta) Command:** Computes the solution to an initial value problem for a differential equation with known partial derivatives.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ list }</td>
<td>{ list }</td>
<td>{ list }</td>
<td>→</td>
<td>{ list }</td>
<td>{ list }</td>
</tr>
<tr>
<td>{ x_{t_0} }</td>
<td>{ x_{t_0} }</td>
<td>{ x_{t_0} }</td>
<td>{ x_{t_0} }</td>
<td>{ x_{t_0} }</td>
<td>{ x_{t_0} }</td>
</tr>
<tr>
<td>{ x_{T_{final}} }</td>
<td>{ x_{T_{final}} }</td>
<td>{ x_{T_{final}} }</td>
<td>{ x_{T_{final}} }</td>
<td>{ x_{T_{final}} }</td>
<td>{ x_{T_{final}} }</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \([\text{SOLVE}] \text{DIFFE RRK} \)

**Affected by Flags:** None

**Remarks:** RRK solves \( y'(t)=f(t,y) \), where \( y(t_0)=y_0 \). The arguments and results are as follows:
\{ \text{list} \} \text{ contains five items in this order:}

- The independent variable \((t)\).
- The solution variable \((y)\).
- The right-hand side of the differential equation (or a variable where the expression is stored).
- The partial derivative of \(y'(t)\) with respect to the solution variable (or a variable where the expression is stored).
- The partial derivative of \(y'(t)\) with respect to the independent variable (or a variable where the expression is stored).

\[ x_{\text{tol}} \] \text{ sets the tolerance value. If a list is used, the first value is the tolerance and the second value is the initial candidate step size.}

\[ x_{\text{T final}} \] \text{ specifies the final value of the independent variable.}

RRK repeatedly calls RKFSTEP as it steps from the initial value to \(x_{\text{T final}}\).

**Example:** Solve the following initial value problem for \(y(8)\), given that \(y(0) = 0\):

\[ y' = \frac{1}{(1 + t^2)} - 2y^2 = f(t, y) \]

The derivative of the function with respect to \(y\) \((\partial f/\partial y)\) is \(-4y\), and the derivative of the function with respect to \(t\) \((\partial f/\partial t)\) is \(-\frac{2t}{(1+t^2)^2}\).

1. Store the independent variable’s initial value, 0, in \(T\).
2. Store the dependent variable’s initial value, 0, in \(Y\).
3. Store the expression, \(\frac{1}{(1+t^2)} - 2y^2\), in \(F\).
4. Store \(\partial f/\partial y\), \(-4y\), in \(FY\).
5. Store \(\partial f/\partial t\), \(-\frac{2t}{(1+t^2)^2}\), in \(FT\).
6. Enter these five items in a list: <\(T \ Y \ F \ FY \ FT\)>.
7. Enter the tolerance. Use estimated decimal place accuracy as a guideline for choosing a tolerance: 0.00001.
8. Enter the final value for the independent variable: 8.
RRK

The stack should look like this:

\[
\begin{pmatrix}
T & Y & F & FY & FT \\
0.00001 & 8
\end{pmatrix}
\]

9. Press \texttt{RRK}. (The calculation takes a moment.) The variable \( T \) now contains 8, and \( Y \) now contains the value 0.123077277659.

The actual answer is 0.123076923077, so the calculated answer has an error of approximately 0.00000035, well within the specified tolerance.

\textbf{Related Commands:} RKF, RKFERR, RKFSTEP, RRKSTEP, RSBERR

RRKSTEP

\textbf{Next Solution Step and Method (RKF or RRK) Command:}
Computes the next solution step \( (h_{\text{next}}) \) to an initial value problem for a differential equation, and displays the method used to arrive at that result.

\[\begin{array}{ccccccc}
\text{Lvl 4} & \text{Lvl 3} & \text{Lvl 2} & \text{Lvl 1} & \rightarrow & \text{Lvl 4} & \text{Lvl 3} & \text{Lvl 2} & \text{Lvl 1} \\
\{ \text{list} \} & x_{\text{to1}} & h & \text{last} & \rightarrow & \{ \text{list} \} & x_{\text{to1}} & h_{\text{next}} & \text{current} \\
\end{array}\]

\textbf{Keyboard Access:} \texttt{SOLVE}(\texttt{DIFFE}) \texttt{RRKS}

\textbf{Affected by Flags:} None

\textbf{Remarks:} The arguments and results are as follows:

- \{ \text{list} \} contains five items in this order:
  - The independent variable \((t)\).
  - The solution variable \((y)\).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of \(y'(t)\) with respect to the solution variable (or a variable where the expression is stored).
The partial derivative of \( y'(t) \) with respect to the independent variable (or a variable where the expression is stored).

- \( x_{tol} \) is the tolerance value.
- \( h \) specifies the initial candidate step.
- \( last \) specifies the last method used (RKF = 1, RRK = 2). If this is the first time you are using RRKSTEP, enter 0.
- \( current \) displays the current method used to arrive at the next step.
- \( h_{next} \) is the next candidate step.

The independent and solution variables must have values stored in them. RRKSTEP steps these variables to the next point upon completion.

Note that the actual step used by RRKSTEP will be less than the input value \( h \) if the global error tolerance is not satisfied by that value. If a stringent global error tolerance forces RRKSTEP to reduce its stepsize to the point that the Runge-Kutta-Fehlberg or Rosenbrock methods fails, then RRKSTEP will use the Euler method to compute the next solution step and will consider the error tolerance satisfied. The Rosenbrock method will fail if the current independent variable is zero and the stepsize \( \leq 2.5 \times 10^{-499} \) or if the variable is nonzero and the stepsize is \( 2.5 \times 10^{-11} \) times its magnitude. The Runge-Kutta-Fehlberg method will fail if the current independent variable is zero and the stepsize \( \leq 1.3 \times 10^{-498} \) or if the variable is nonzero and the stepsize is \( 1.3 \times 10^{-10} \) times its magnitude.

**Related Commands:** RKF, RKFERR, RKFSTEP, RRK, RSBERR
**RSBERR**

**Error Estimate for Rosenbrock Method Command:** Returns an error estimate for a given step $h$ when solving an initial values problem for a differential equation.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ list }</code></td>
<td>$h$</td>
<td>→</td>
<td><code>{ list }</code></td>
<td>$h$</td>
<td>$y_{\Delta}$</td>
<td>error</td>
</tr>
</tbody>
</table>

**Keyboard Access:** 🕵️‍♂️(SOLVE) DIFFE RSBER

**Affected by Flags:** None

**Remarks:** The arguments and results are as follows:

- `{ list }` contains five items in this order:
  - The independent variable ($t$).
  - The solution variable ($y$).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of $y'(t)$ with respect to the solution variable (or a variable where the expression is stored).
  - The partial derivative of $y'(t)$ with respect to the independent variable (or a variable where the expression is stored).
- $h$ is a real number that specifies the initial step.
- $y_{\Delta}$ displays the change in solution.
- `error` displays the absolute error for that step. The absolute error is the absolute value of the estimated error for a scalar problem, and the row (infinity) norm of the estimated error vector for a vector problem. (The latter is a bound on the maximum error of any component of the solution.) A zero error indicates that the Rosenbrock method failed and Euler's method was used instead.

**Related Commands:** RKF, RKFERR, RKFSTEP, RRK, RRKSTEP
**Residual Command:** Computes the residual $B - AZ$ of the arrays $B$, $A$, and $Z$.  

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\text{vector}]_B$</td>
<td>$[[\text{matrix}]_A]$</td>
<td>$[\text{vector}]_Z$</td>
<td>$\rightarrow$</td>
<td>$[\text{vector}]_{B-AZ}$</td>
</tr>
<tr>
<td>$[[\text{matrix}]_B]$</td>
<td>$[[\text{matrix}]_A]$</td>
<td>$[[\text{matrix}]_Z]$</td>
<td>$\rightarrow$</td>
<td>$[[\text{matrix}]_{B-AZ}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [MATR] [NXT] [RSD]

**Affected by Flags:** None

**Remarks:** $A$, $B$, and $Z$ are restricted as follows:

- $A$ must be a matrix.
- The number of columns of $A$ must equal the number of elements of $Z$ if $Z$ is a vector, or the number of rows of $Z$ if $Z$ is a matrix.
- The number of rows of $A$ must equal the number of elements of $B$ if $B$ is a vector, or the number of rows of $B$ if $B$ is a matrix.
- $B$ and $Z$ must both be vectors or both be matrices.
- $B$ and $Z$ must have the same number of columns if they are matrices.

RSD is typically used for computing a correction to $Z$, where $Z$ has been obtained as an approximation to the solution $X$ to the system of equations $AX = B$.  

*Command Reference 3-291*
**RSWP**

**Row Swap Command:** Swaps rows \(i\) and \(j\) of a matrix and returns the modified matrix, or swaps elements \(i\) and \(j\) of a vector and returns the modified vector.

\[
\begin{array}{c|c|c|c|c}
\text{Level 3} & \text{Level 2} & \text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
[[\text{matrix}]]_1 & n_{rowi} & n_{rowj} & \rightarrow & [[\text{matrix}]]_2 \\
[\text{vector}]_1 & n_{elementi} & n_{elementj} & \rightarrow & [\text{vector}]_2 \\
\end{array}
\]

**Keyboard Access:** (MTH) MATR \(\text{ROW NXT} \) RSWP

**Affected by Flags:** None

**Remarks:** Row numbers are rounded to the nearest integer. Vector arguments are treated as column vectors.

**Related Commands:** CSWP, ROW+, ROW-

---

**R→B**

**Real to Binary Command:** Converts a positive real integer to its binary integer equivalent.

\[
\begin{array}{c|c|c}
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
n & \rightarrow & \#n \\
\end{array}
\]

**Keyboard Access:** (MTH) BASE \(\text{R→B} \)

**Affected by Flags:** Binary Integer Wordsize \((-5\) through \(-10\)), Binary Integer Base \((-11, -12)\)
Remarks: For any value of \( n \leq 0 \), the result is \# 0. For any value of \( n > 1 \). 84467440738E19 (base 10), the result is \# FFFFFFFFFFFFFFFFF (base 16).

Related Commands: B→R

---

**R→C**

**Real to Complex Command:** Combines two real numbers or real arrays into a single complex number or complex array.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>→</td>
<td>(x,y)</td>
</tr>
<tr>
<td>[ R-array_1 ]</td>
<td>[ R-array_2 ]</td>
<td>→</td>
<td>[ C-array ]</td>
</tr>
</tbody>
</table>

Keyboard Access:

\[
\begin{align*}
\text{[PRG]} & \quad \text{TYPE} & \quad \text{NXT} & \quad \text{R→C} \\
\text{[MTH]} & \quad \text{NXT} & \quad \text{CMPL} & \quad \text{R→C}
\end{align*}
\]

Affected by Flags: None

Remarks: The level 2 argument represents the real element(s) of the complex result. The level 1 argument represents the imaginary element(s) of the complex result.

Array arguments must have the same dimensions.

Related Commands: C→R, IM, RE
\[ R \rightarrow D \]

**Radians to Degrees Function:** Converts a real number expressed in radians to its equivalent in degrees.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>→</td>
<td>( \frac{180}{\pi} x )</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'R→D(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH} \ \text{REAL} \ \text{NXT} \ \text{NXT} \ \text{R→D} \)

**Affected by Flags:** Numerical Results (-3)

**Remarks:** This function operates independently of the angle mode.

**Related Commands:** D→R

---

**SAME**

**Same Object Command:** Compares two objects, and returns a true result (1) if they are identical, and a false result (0) if they are not.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( obj_1 )</td>
<td>( obj_2 )</td>
<td>→</td>
<td>0/1</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{PRG} \ \text{TEST} \ \text{NXT} \ \text{SAME} \)

**Affected by Flags:** None

**Remarks:** SAME is identical in effect to \( == \) for all object types except algebraics, names, and some units. (For algebraics and names, \( == \) returns an expression that can be evaluated to produce a test result based on numerical values.)

**Examples:** \( \{ A \ B \} (4,5) \) SAME returns \( 0 \).
SCALE

(A B) (B A) SAME returns 0.
"CATS" "CATS" SAME returns 1.

Related Commands: TYPE, ==

SBRK

Serial Break Command: Interrupts serial transmission or reception.

Keyboard Access: \(\leftarrow\) I/O NXT SERIA SBRK

Affected by Flags: I/O Device (–33)

Remarks: SBRK is typically used when a problem occurs in a serial data transmission.

Related Commands: BUFLEN, SRECV, STIME, XMIT

SCALE

Scale Plot Command: Adjusts the first two parameters in PPAR, \(<x_{\text{min}}, y_{\text{min}}>, \text{ and } <x_{\text{max}}, y_{\text{max}}>,\) so that \(x_{\text{scale}}\) and \(y_{\text{scale}}\) are the new plot horizontal and vertical scales, and the center point doesn’t change.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>—</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_{\text{scale}})</td>
<td>(y_{\text{scale}})</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \(\leftarrow\) (PLOT) PPAR NXT SCALE

Affected by Flags: None

Remarks: The scale in either direction is the number of user units per tick mark. The default scale in both directions is 1 user unit per tick mark.
SCATR PLOT

**Draw Scatter Plot Command:** Draws a scatterplot of \((x, y)\) data points from the specified columns of the current statistics matrix (reserved variable \(\Sigma DAT\)).

**Keyboard Access:** \(\text{\(\downarrow\text{STAT}\) \(\text{\textbf{PLOT}}\) \(\text{\textbf{SCATR}}\)}\)

**Affected by Flags:** None

**Remarks:** The data columns plotted are specified by XCOL and YCOL, and are stored as the first two parameters in the reserved variable \(\Sigma PAR\). If no data columns are specified, columns 1 (independent) and 2 (dependent) are selected by default. The \(y\)-axis is autoscaled and the plot type is set to SCATTER.

When SCATR PLOT is executed from a program, the resulting display does not persist unless PICTURE or PVIEW is subsequently executed.

If PICTURE is subsequently executed, pressing \(\text{\(\downarrow\text{STAT}\)}\) in the Picture environment draws a line to fit the data using the currently specified statistical model.

**Example:** The following program plots a scatter plot of the data in columns 3 and 4 of \(\Sigma DAT\), draws a best fit line, and displays the plot:

```
< 3 XCOL 4 YCOL SCATR PLOT BESTFIT SPLINE STEQ
FUNCTION DRAW \(\# \# 0d \# \# 0d \) PVIEW 7 FREEZE >
```

**Related Commands:** BARPLOT, PICTURE, HISTPLOT, PVIEW, SCL\(\Sigma\), XCOL, YCOL
SCATTER

Scatter Plot Type Command: Sets the plot type to SCATTER.

Keyboard Access: \texttt{\textasciitilde PLOT NXT STAT PTYPE SCATT}

Affected by Flags: None

Remarks: When the plot type is SCATTER, the DRAW command plots points by obtaining $x$ and $y$ coordinates from two columns of the current statistics matrix (reserved variable \texttt{Z DAT}). The columns are specified by the first and second parameters in the reserved variable \texttt{PAR} (using the XCOL and YCOL commands). The plotting parameters are specified in the reserved variable \texttt{PPAR}, which has this form:

\[
\{ \langle x_{\min}, y_{\min} \rangle, \langle x_{\max}, y_{\max} \rangle, \text{indep}, \text{res}, \text{axes}, \text{ptype}, \text{depend} \} \]

For plot type SCATTER, the elements of \texttt{PPAR} are used as follows:

- \(\langle x_{\min}, y_{\min} \rangle\) is a complex number specifying the lower left corner of \texttt{PICT} (the lower left corner of the display range). The default value is \((-6.5, -3.1)\).
- \(\langle x_{\max}, y_{\max} \rangle\) is a complex number specifying the upper right corner of \texttt{PICT} (the upper right corner of the display range). The default value is \((6.5, 3.2)\).
- \texttt{indep} is a name specifying the independent variable. The default value of \texttt{indep} is \texttt{X}.
- \texttt{res} is not used.
- \texttt{axes} is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is \(\emptyset, \emptyset\).
- \texttt{ptype} is a command name specifying the plot type. Executing the command SCATTER places the name SCATTER in \texttt{ptype}.
- \texttt{depend} is a name specifying the dependent variable. The default value is \texttt{Y}.

Related Commands: \texttt{BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE,}
SCATTER
PCONTOUR, POLAR, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

SCHUR

Schur Decomposition of a Square Matrix Command: Returns the Schur decomposition of a square matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[[\text{matrix }]_A]$</td>
<td>$\rightarrow$</td>
<td>$[[\text{matrix }]_Q]$</td>
<td>$[[\text{matrix }]_T]$</td>
</tr>
</tbody>
</table>

Keyboard Access: MATHERMATFACTRSCHUR

Affected by Flags: None

Remarks: SCHUR decomposes $A$ into two matrices $Q$ and $T$:
- If $A$ is a complex matrix, $Q$ is a unitary matrix, and $T$ is an upper-triangular matrix.
- If $A$ is a real matrix, $Q$ is an orthogonal matrix, and $T$ is an upper quasi-triangular matrix ($T$ is upper block triangular with $1 \times 1$ or $2 \times 2$ diagonal blocks where the $2 \times 2$ blocks have complex conjugate eigenvalues).

In either case, $A \cong Q \times T \times \text{TRN}(Q)$.

Related Commands: LQ, LU, QR, SVD, SVL, TRN
**SCI**

**Scientific Mode Command:** Sets the number display format to Scientific mode, which displays one digit to the left of the fraction mark and \( n \) significant digits to the right.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \leftarrow \text{(MODES)} \text{ FMT SCI} \)

**Affected by Flags:** None

**Remarks:** Scientific mode is equivalent to scientific notation using \( n + 1 \) significant digits, where \( 0 \leq n \leq 11 \). (Values for \( n \) outside this range are rounded to the nearest integer.) In Scientific mode, numbers are displayed and printed like this:

\[
(\text{sign}) \text{ mantissa } \times 10^{(\text{sign}) \text{ exponent}}
\]

where the mantissa has the form \( n.(n \ldots) \) and has zero to 11 decimal places, and the exponent has one to three digits.

**Example:** The number 103.6 in Scientific mode to four decimal places appears as \( 1.0360 \times 10^2 \).

**Related Commands:** ENG, FIX, STD

---

**SCLΣ**

**Scale Sigma Command:** Adjusts \( (x_{\min}, y_{\min}) \) and \( (x_{\max}, y_{\max}) \) in \( PPAR \) so that a subsequent scatter plot exactly fills \( PICT \).

**Keyboard Access:** None. Must be typed in.

**Affected by Flags:** None

**Remarks:** When the plot type is SCATTER, the command AUTO incorporates the functions of SCLΣ. In addition, the command
SCL\Sigma

SCATR PLOT automatically executes AUTO to achieve the same result. SCL\Sigma is included in the HP 48 for compatibility with the HP 28.

**Related Commands:** AUTO, SCATR PLOT

---

SCONJ

**Store Conjugate Command:** Conjugates the contents of a named object.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** (+)(MEMORY) ARITH (NXT) SCON

**Affected by Flags:** None

**Remarks:** The named object must be a number, an array, or an algebraic object. For information on conjugation, see CONJ.

**Related Commands:** CONJ, SINV, SNEG

---

SDEV

**Standard Deviation Command:** Calculates the sample standard deviation of each of the $m$ columns of coordinate values in the current statistics matrix (reserved variable $\Sigma DAT$).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>$x_{sdev}$</td>
</tr>
<tr>
<td></td>
<td>→</td>
<td>$[ x_{sdev1} x_{sdev2} \ldots x_{sdevm} ]$</td>
</tr>
</tbody>
</table>
Keyboard Access:  

`STAT 1 VAR SDEV`

Affected by Flags:  None

Remarks:  SDEV returns a vector of \( m \) real numbers, or a single real number if \( m = 1 \). The standard deviations (the square root of the variances) are computed using this formula:

\[
\sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}
\]

where \( x_i \) is the \( i \)th coordinate value in a column, \( \bar{x} \) is the mean of the data in this column, and \( n \) is the number of data points.

Related Commands:  MAX\( \Sigma \), MEAN, MIN\( \Sigma \), PSDEV, PVAR, TOT, VAR

---

Send Object Command: Sends a copy of the named objects to a Kermit device.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ name&lt;sub&gt;1&lt;/sub&gt; ... name&lt;sub&gt;n&lt;/sub&gt; }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{{ name&lt;sub&gt;old&lt;/sub&gt; name&lt;sub&gt;new&lt;/sub&gt; } name ... }</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access:  

`I/O SEND`

Affected by Flags:  I/O Device (−33), I/O Data Format (−35), I/O Messages (−39)

If flag −35 is clear (ASCII transfer), the translation setting also has an effect.

Remarks:  Data is always sent from a local Kermit, but can be sent either to another local Kermit (which must execute RECV or RECN) or to a server Kermit.
**SEND**

To rename an object when sending it, include the old and new names in an embedded list.

**Examples:** Executing `{ { AAA BBB } } SEND` sends the variable named `AAA` but changes its name to `BBB`.

Executing `{ { AAA BBB } CCC } SEND` sends `AAA` as `BBB` and sends `CCC` under its own name. (If the new name is not legal on the HP 48, just enter it as a string.)

**Related Commands:** `BAUD`, `CLOSEIO`, `CKSM`, `FINISH`, `KERRM`, `KGET`, `PARITY`, `RECN`, `RECV`, `SERVER`, `TRANSIO`

---

**SEQ**

**Sequential Calculation Command:** Returns a list of results generated by repeatedly executing `objexec` using `index` over the range `x_{start}` to `x_{end}`, in increments of `x_{incr}`.

<table>
<thead>
<tr>
<th>Level 5</th>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objexec</code></td>
<td><code>index</code></td>
<td><code>x_{start}</code></td>
<td><code>x_{end}</code></td>
<td><code>x_{incr}</code></td>
<td>→</td>
<td><code>{ list }</code></td>
</tr>
</tbody>
</table>

**Keyboard Access:** `PRG` `LIST` `PROC` `NXT` `SEQ`

**Affected by Flags:** None

**Remarks:** `objexec` is nominally a program or algebraic object that is a function of `index`, but can actually be any object. `index` must be a global or local name. The remaining objects can be anything that will evaluate to real numbers.

The action of SEQ for arbitrary inputs can be predicted exactly from this equivalent program:

```
x_{start} x_{end} FOR index objexec EVAL x_{incr} STEP n → LIST
```

where `n` is the number of new objects left on the stack by the `FOR . . . STEP` loop. Notice that `index` becomes a local variable regardless of its original type.
Example: \( n^2 \) \( n \) 1 4 1 returns \( \{ 1 \ 4 \ 9 \ 16 \ 3 \} \).

Related Commands: DOSUBS, STREAM

---

**SERVER**

**Server Mode Command:** Selects Kermit Server mode.

**Keyboard Access:**

\[ \text{Q}{/0} \] \( \text{SRVR} \) \( \text{SERVE} \)

**Affected by Flags:** I/O Device (−33), I/O Data Format (−35), RECV Overwrite (−36), I/O Messages (−39)

**Remarks:** A Kermit server (a Kermit device in Server mode) passively processes requests sent to it by the local Kermit. The server receives data in response to SEND, transmits data in response to KGET, terminates Server mode in response to FINISH or LOGOUT, and transmits a directory listing in response to a generic directory request.

Server mode supports Kermit Host Command packets. This allows you, for instance, to use a PC to type into the HP 48’s command line. (This is especially convenient while testing programs.) Do this as follows:

1. Set up the HP 48 for data transfer to a computer, as described in “Transferring Data Between the HP 48 and a Computer” in chapter 27 of the *HP 48 User’s Guide*.

2. Execute SERVER to set the HP 48 to Server mode.

3. On the PC, type REMOTE HOST followed by up to 89 characters to be entered into the HP 48 command line.

4. Press \( \text{Return} \) to transmit and execute the commands. The HP 48 executes the transmitted commands, then sends back to the PC’s display the resulting contents of the stack as the HP 48 would normally display them.

Command Reference 3-303
If you use a PC to write programs for the HP 48, you should include the `%%HP...;` header in the program. See the discussion of ASCII mode in chapter 27 of the *HP 48 User’s Guide*.

**Related Commands:** BAUD, CKSM, FINISH, KERRM, KGET, PARITY, PKT, RECN, RECV, SEND, TRANSIO

---

**SF**

**Set Flag Command:** Sets a specified user or system flag.

```
{ }
```

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{flag\ number} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- [PRG] TEST [NXT] [NXT] SF
- [MODES] FLAG SF

**Affected by Flags:** None

**Remarks:** User flags are numbered 1 through 64. System flags are numbered \(-1\) through \(-64\). See appendix C, “System Flags,” for a listing of HP 48 system flags and their flag numbers.

**Related Commands:** CF, FC?, FC?C, FS?, FS?C
SHOW

Show Variable Command: Returns 'symb₂', which is equivalent to 'symb₁' except that all implicit references to a variable name are made explicit.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb₁'</td>
<td>'name'</td>
<td>→</td>
<td>'symb₂'</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>{ name₁ name₂ ... }</td>
<td>→</td>
<td>'symb₂'</td>
</tr>
</tbody>
</table>

Keyboard Access:  

Affected by Flags: Numerical Results (-3)

Remarks: If the level 1 argument is a list, SHOW evaluates all global variables in 'symb₁' not contained in the list.

Example: If 7 is stored in C and 5 is stored in D, then

\[ \text{LW Y Z SHOM} \]

returns '-1 d-

Related Commands: COLCT, EXPAN, ISOL, QUAD

SIDENS

Silicon Intrinsic Density Command: Calculates the intrinsic density of silicon as a function of temperature, \( x_T \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_T )</td>
<td>→</td>
<td>( x_{\text{density}} )</td>
</tr>
<tr>
<td>( x_{\text{unit}} )</td>
<td>→</td>
<td>( x_{1/cm^3} )</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'SIDENS(symb)'</td>
</tr>
</tbody>
</table>
**SIDENS**

**Keyboard Access:** UTILS SIDEN

**Affected by Flags:** Numerical Results (-3)

**Remarks:** If $x_T$ is a unit object, it must reduce to a pure temperature, and the density is returned as a unit object with units of 1/cm$^3$.

If $x_T$ is a real number, its units are assumed to be K, and the density is returned as a real number with implied units of 1/cm$^3$.

$x_T$ must be between 0 and 1685 K.

---

**SIGN**

**Sign Function:** Returns the sign of a real number argument, the sign of the numerical part of a unit object argument, or the unit vector in the direction of a complex number argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>→</td>
<td>$z_2$</td>
</tr>
<tr>
<td>$x_{\text{unit}}$</td>
<td>→</td>
<td>$x_{\text{sign}}$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'SIGN(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- **MTH REAL** (returns the sign of a number)
- **MTH** (returns the unit vector of a complex number)

**Affected by Flags:** Numerical Results (-3)

**Remarks:** For real number and unit object arguments, the sign is defined as +1 for positive arguments, −1 for negative arguments, and 0 for argument 0.

For a complex argument:
\[
\text{SIGN}(x + iy) = \frac{x}{\sqrt{x^2 + y^2}} + \frac{iy}{\sqrt{x^2 + y^2}}
\]

**Examples:**
32 \_ft \_SIGN returns 1.

(1,1) \_SIGN returns (0.707106781187, 0.707106781187).

**Related Commands:** ABS, MANT, XPON

---

**SIMU**

**Simultaneous Plotting Command:** Enables and disables simultaneous plotting.

**Keyboard Access:** \(_p(\text{PLOT}) \text{NXT FLAG SIMU}_p\)

**Affected by Flags:** Simultaneous Plotting (–28)

**Remarks:** \_SIMU changes to \_SIMU when flag –28 is enabled (and simultaneous plotting is enabled).

If the calculator is in program entry mode, pressing the menu key echoes AXES, CNCT, and SIMU flag numbers to the command line. Pressing \(_p\_p_\) or \(_p_\) first echoes the flag numbers and SF or CF to the command line.

**Related Commands:** AXES, CF, SF

---

**SIN**

**Sine Analytic Function:** Returns the sine of the argument.

---

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>_z _</td>
<td>→</td>
<td>_sin _z</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'SIN(symb)'</td>
</tr>
<tr>
<td>_x_unit_angular _</td>
<td>→</td>
<td>sin (_x_unit_angular)</td>
</tr>
</tbody>
</table>
SIN

Keyboard Access:  \textbf{SIN}

Affected by Flags:  Numerical Results \((-3\), Angle Mode \((-17, -18\))

Remarks:  For real arguments, the current angle mode determines the number's units, unless angular units are specified.

For complex arguments, \(\sin(x + iy) = \sin x \cosh y + i \cos x \sinh y\).

If the argument for SIN is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing SIN with a unit object, the angle mode must be set to Radians (since this is a “neutral” mode).

Related Commands:  ASIN, COS, TAN

\newpage

\textbf{SINH}

Hyperbolic Sine Analytic Function:  Returns the hyperbolic sine of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z)</td>
<td>→</td>
<td>(\sinh z)</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'SINH(symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access:  \textbf{MTH HYP SINH}

Affected by Flags:  Numerical Results \((-3\))

Remarks:  For complex arguments, \(\sinh(x + iy) = \sinh x \cos y + i \cosh x \sin y\).

Related Commands:  ASINH, COSH, TANH
SINV

Store Inverse Command: Replaces the contents of the named variable with its inverse.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \(\text{ MEMORY ARITH NXT SINV}\)

Affected by Flags: None

Remarks: The named object must be a number, a matrix, an algebraic object, or a unit object. For information on reciprocals, see INV.

Related Commands: INV, SCONJ, SNEG

SIZE

Size Command: Returns the number of characters in a string, the number of elements in a list, the dimensions of an array, the number of objects in a unit object or algebraic object, or the dimensions of a graphics object.
SIZE

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;string&quot;</td>
<td>→</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>{ list }</td>
<td>→</td>
<td>{ n }</td>
<td></td>
</tr>
<tr>
<td>[ vector ]</td>
<td>→</td>
<td></td>
<td>{ n m }</td>
</tr>
<tr>
<td>[[ matrix ]]</td>
<td>→</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grob</td>
<td>→</td>
<td>#n_width</td>
<td>#m_height</td>
</tr>
<tr>
<td>PICT</td>
<td>→</td>
<td>#n_width</td>
<td>#m_height</td>
</tr>
<tr>
<td>x_unit</td>
<td>→</td>
<td></td>
<td>n</td>
</tr>
</tbody>
</table>

Keyboard Access:

- (B) (CHARS) EIZE
- PRG LIST ELEM SIZE
- PRG GROB (NXT) SIZE

Affected by Flags: None

Remarks: The size of a unit is computed as follows: the scalar (+1), the underscore (+1), each unit name (+1), operator or exponent (+1), and each prefix (+2).

Any object type not listed above returns a value of 1.

Related Commands: CHR, NUM, POS, REPL, SUB

SL

Shift Left Command: Shifts a binary integer one bit to the left.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n_1</td>
<td>→</td>
<td>#n_2</td>
</tr>
</tbody>
</table>
SLB

Keyboard Access: BEASE

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The most significant bit is shifted out to the left and lost, while the least significant bit is regenerated as a zero. SL is equivalent to binary multiplication by 2, truncated to the current wordsize.

Related Commands: ASR, SLB, SR, SRB

---

SLB

Shift Left Byte Command: Shifts a binary integer one byte to the left.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
</tbody>
</table>

Keyboard Access: BEASE

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The most significant byte is shifted out to the left and lost, while the least significant byte is regenerated as zero. SLB is equivalent to binary multiplication by $2^8$ (or executing SL eight times), truncated to the current wordsize.

Related Commands: ASR, SL, SR, SRB
SLOPEFIELD

SLOPEFIELD Plot Type Command: Sets the plot type to SLOPEFIELD.

Keyboard Access:  \( \text{←} \) \text{PLOT} \text{ Nxt} \ 3D \text{ PTYPE SLOPE} \)

Affected by Flags: None

Remarks: When plot type is set to SLOPEFIELD, the DRAW command plots a slope representation of a scalar function with two variables. SLOPEFIELD requires values in the reserved variables \( EQ, VPAR, \) and \( PPAR \).

\( VPAR \) has the following form:

\[
\begin{align*}
\xi & x_{\text{left}} x_{\text{right}} y_{\text{near}} y_{\text{far}} z_{\text{low}} z_{\text{high}} x_{\text{min}} x_{\text{max}} y_{\text{min}} y_{\text{max}} x_{\text{eye}} y_{\text{eye}} z_{\text{eye}} x_{\text{step}} y_{\text{step}} \\
\end{align*}
\]

For plot type SLOPEFIELD, the elements of \( VPAR \) are used as follows:

- \( x_{\text{left}} \) and \( x_{\text{right}} \) are real numbers that specify the width of the view space.
- \( y_{\text{near}} \) and \( y_{\text{far}} \) are real numbers that specify the depth of the view space.
- \( z_{\text{low}} \) and \( z_{\text{high}} \) are real numbers that specify the height of the view space.
- \( x_{\text{min}} \) and \( x_{\text{max}} \) are not used.
- \( y_{\text{min}} \) and \( y_{\text{max}} \) are not used.
- \( x_{\text{eye}}, y_{\text{eye}}, \) and \( z_{\text{eye}} \) are real numbers that specify the point in space from which the graph is viewed.
- \( x_{\text{step}} \) and \( y_{\text{step}} \) are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable \( PPAR \), which has this form:

\[
\begin{align*}
\xi & (x_{\text{min}}, y_{\text{min}}) (x_{\text{max}}, y_{\text{max}}) \text{ indep res axes ptype depend} \\
\end{align*}
\]

For plot type SLOPEFIELD, the elements of \( PPAR \) are used as follows:

- \( (x_{\text{min}}, y_{\text{min}}) \) is not used.
SNEG

Store Negate Command: Replaces the contents of a variable with its negative.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access:  

Affected by Flags: None

Remarks: The named object must be a number, an array, an algebraic object, a unit object, or a graphics object. For information on negation, see NEG.

Related Commands: NEG, SCONJ, SINV
**SNRM**

**Spectral Norm Command:** Returns the spectral norm of an array.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ array ]</td>
<td>→</td>
<td>( x_{\text{spectral norm}} )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH MATR NORM SNRM

**Affected by Flags:** None

**Remarks:** The spectral norm of a vector is its Euclidean length, and is equal to the largest singular value of a matrix.

**Related Commands:** ABS, CNRM, COND, RNRM, SRAD, TRACE

---

**SOLVEQN**

**Start Equation Solver Command:** Starts the appropriate solver for a specified set of equations.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( m )</td>
<td>( 0/1 )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** EQ LIB SOLVE

**Affected by Flags:** Unit Type (60), Units Usage (61)

**Remarks:** SOLVEQN sets up and starts the appropriate solver for the specified set of equations, bypassing the Equation Library catalogs. It sets \( EQ \) (and \( Mpar \) if more than one equation is being solved), sets the unit options according to flags 60 and 61, and starts the appropriate solver.
SOLVEQN uses subject and title numbers (levels 3 and 2) and a “PICT” option (level 1), and returns nothing. Subject and title numbers are listed in chapter 4. If the “PICT” option is 0, PICT is not affected; otherwise, the equation picture is copied into PICT.

Related Commands: EQNLIB, MSOLVER

---

**SORT**

**Ascending Order Sort Command:** Sorts the elements in a list in ascending order.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ list }_1</td>
<td>→</td>
<td>{ list }_2</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- \texttt{PRG LIST PROC NXT SORT}
- \texttt{MTH LIST SORT}

**Affected by Flags:** None

**Remarks:** The elements in the list can be real numbers, strings, lists, names, binary integers, or unit objects. However, all elements in the list must all be of the same type. Strings and names are sorted by character code number. Lists of lists are sorted by the first element in each list.

To sort in reverse order, use \texttt{SORT REVLIST}.

**Related Commands:** REVLIST
SPHERE

Spherical Mode Command: Sets Spherical coordinate mode.

Keyboard Access:

- `MTH VECTR [NXT] SPHER`
- `MODES ANGL SPHER`

Affected by Flags: None

Remarks: SPHERE sets flags $-15$ and $-16$, and displays the Θ annunciation.

In Spherical mode, vectors are displayed as polar components. Therefore, a 3D vector would appear as $[r \Delta \theta \Delta \phi]$.

Related Commands: CYLIN, RECT

SQ

Square Analytic Function: Returns the square of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>→</td>
<td>$z^2$</td>
</tr>
<tr>
<td>$x_unit$</td>
<td>→</td>
<td>$x^2 _unit^2$</td>
</tr>
<tr>
<td>$[[\text{matrix}]]$</td>
<td>→</td>
<td>$[[\text{matrix} \times \text{matrix}]]$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'SQ(symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access: 

- $(x^2)$

Affected by Flags: Numerical Results ($-3$)

Remarks: The square of a complex argument $(x, y)$ is the complex number $(x^2 - y^2, 2xy)$.

Matrix arguments must be square.

Related Commands: $\sqrt{}$, $^\wedge$
SR

Shift Right Command: Shifts a binary integer one bit to the right.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
</tbody>
</table>

Keyboard Access: MTH BASE NXT BIT SR

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The least significant bit is shifted out to the right and lost, while the most significant bit is regenerated as a zero. SR is equivalent to binary division by 2.

Related Commands: ASR, SL, SLB, SRB

SRAD

Spectral Radius Command: Returns the spectral radius of a square matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[ matrix ]]ₙₓₙ</td>
<td>→</td>
<td>x_{spectral radius}</td>
</tr>
</tbody>
</table>

Keyboard Access: MTH MATR NORM SRAD

Affected by Flags: None

Remarks: The spectral radius of a matrix is a measure of the size of the matrix, and is equal to the absolute value of the largest eigenvalue of the matrix.
SRAD

Related Commands: COND, SNRM, TRACE

SRB

Shift Right Byte Command: Shifts a binary integer one byte to the right.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n₁</td>
<td>→</td>
<td>#n₂</td>
</tr>
</tbody>
</table>

Keyboard Access: MTH BASE NXT BYTE SRB

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The least significant byte is shifted out to the right and lost, while the most significant byte is regenerated as zero. SRB is equivalent to binary division by 2^8 (or executing SR eight times).

Related Commands: ASR, SL, SLB, SR

SRECV

Serial Receive Command: Reads up to n characters from the serial input buffer and returns them as a string, along with a digit indicating whether errors occurred.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>→</td>
<td>'string'</td>
<td>0/1</td>
</tr>
</tbody>
</table>

Keyboard Access: I/O NXT SERIA SRECV
Affected by Flags:  I/O Device (–33)

Remarks:  SRECV does not use Kermit protocol.

If \( n \) characters are not received within the time specified by STIME (default is 10 seconds), SRECV “times out”, returning a \& to level 1 and as many characters as were received to level 2.

If the level 2 output from BUFLEN is used as the input for SRECV, SRECV will not have to wait for more characters to be received. Instead, it returns the characters already in the input buffer.

If you want to accumulate bytes in the input buffer before executing SRECV, you must first open the port using OPENIO (if the port isn’t already open).

SRECV can detect three types of error when reading the input buffer:

- Framing errors and UART overruns (both causing "Receive Error" in ERRM).
- Input-buffer overflows (causing "Receive Buffer Overflow" in ERRM).
- Parity errors (causing "Parity Error" in ERRM).

SRECV returns \& if an error is detected when reading the input buffer, or 1 if no error is detected.

Parity errors do not stop data flow into the input buffer. However, if a parity error occurs, SRECV stops reading data after encountering a character with an error.

Framing, overrun, and overflow errors cause all subsequently received characters to be ignored until the error is cleared. SRECV does not detect and clear any of these types of errors until it tries to read the byte where the error occurred. Since these three errors cause the byte where the error occurred and all subsequent bytes to be ignored, the input buffer will be empty after all previously received good bytes have been read. Therefore, SRECV detects and clears these errors when it tries to read a byte from an empty input buffer.

Note that BUFLEN also clears the above-mentioned framing, overrun, and overflow errors. Therefore, SRECV cannot detect an input-buffer overflow after BUFLEN is executed, unless more characters were received after BUFLEN was executed (causing the input buffer to overflow again). SRECV also cannot detect framing and UART overrun errors cleared by BUFLEN. To find where the data error
SRECV

occurred, save the number of characters returned by BUFLEN (which gives the number of “good” characters received), because as soon as the error is cleared, new characters can enter the input buffer.

**Example:** If 10 good bytes were received followed by a framing error, then an SRECV command told to read 10 bytes would *not* indicate an error. Only when SRECV tries to read the byte that caused the framing error does it return a Ø. Similarly, if the input buffer overflowed, SRECV would not indicate an error until it tried to read the first byte that was lost due to the overflow.

**Related Commands:** BUFLEN, CLOSEIO, OPENIO, SBRK, STIME, XMIT

---

**SST**

**Execute Program Step Operation:** Returns and executes the next step of a program. If the next step is a subroutine, executes the subroutine in a single step.

**Keyboard Access:**

| PRG | NXT | RUN | SST |

**Affected by Flags:** None

**Remarks:** SST is not programmable.

**Related Commands:** NEXT (operation), SST|
START

START Definite Loop Structure Command: Begins START ... NEXT and START ... STEP definite loop structures.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>$x_{\text{start}}$</td>
<td>$x_{\text{finish}}$</td>
<td>→</td>
</tr>
<tr>
<td>NEXT</td>
<td></td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>STEP</td>
<td>$x_{\text{increment}}$</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>STEP</td>
<td>'symb_{\text{increment}}'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \texttt{PRG} \texttt{BRCH} \texttt{START} \texttt{START}

Affected by Flags: None

Remarks: Definite loop structures execute a command or sequence of commands a specified number of times.

- START ... NEXT executes a portion of a program a specified number of times. The syntax is this:
  \[ x_{\text{start}} \ x_{\text{finish}} \text{ START loop-clause NEXT} \]
  START takes two numbers ($x_{\text{start}}$ and $x_{\text{finish}}$) from the stack and stores them as the starting and ending values for a loop counter. Then the loop clause is executed. NEXT increments the counter by 1 and tests to see if its value is less than or equal to $x_{\text{finish}}$. If so, the loop clause is executed again. Notice that the loop clause is always executed at least once.

- START ... STEP works just like START ... NEXT, except that it can use an increment value other than 1. The syntax is this:
  \[ x_{\text{start}} \ x_{\text{finish}} \text{ START loop-clause increment STEP} \]
  START takes two numbers ($x_{\text{start}}$ and $x_{\text{finish}}$) from the stack and stores them as the starting and ending values of the loop counter. Then the loop clause is executed. STEP takes $x_{\text{increment}}$ from the stack and increments the counter by that value. If the argument of STEP is an algebraic or a name, it is automatically evaluated to a number.
The increment value can be positive or negative:

- If positive, the loop is executed again when the counter is less than or equal to $x_{\text{finish}}$.
- If negative, the loop is executed when the counter is greater than or equal to $x_{\text{finish}}$.

**Related Commands:** FOR, NEXT, STEP

---

**STD**

**Standard Mode Command:** Sets the number display format to Standard mode.

**Keyboard Access:** (€)(MODES) FMT STD

**Affected by Flags:** None

**Remarks:** Executing STD has the same effect as clearing flags $-49$ and $-50$.

Standard format (ANSI Minimal BASIC Standard X3J2) produces the following results when displaying or printing a number:

- Numbers that can be represented exactly as integers with 12 or fewer digits are displayed without a fraction mark or exponent. Zero is displayed as $0$.
- Numbers that can be represented exactly with 12 or fewer digits, but not as integers, are displayed with a fraction mark but no exponent. Leading zeros to the left of the fraction mark and trailing zeros to the right of the fraction mark are omitted.
- All other numbers are displayed in scientific notation (see SCI) with both a fraction mark (with one number to the left) and an exponent (of one to three digits). There are no leading or trailing zeros.

In algebraic objects, integers less than $10^3$ are always displayed in Standard mode.

**Example:** The following table provides examples of numbers displayed in Standard mode:
### Related Commands:
ENG, FIX, SCI

### STEP

**STEP Command:** Defines the increment (step) value, and ends definite loop structure.

See the FOR and START command entries for syntax information.

**Keyboard Access:**

PRG BRCH FOR STEP

PRG BRCH START STEP

**Remarks:** See the FOR and START keyword entries for more information.

**Related Commands:** FOR, NEXT, START
**STEQ**

**Store in EQ Command:** Stores an object into the reserved variable `EQ` in the current directory.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>obj</code></td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** This command must be typed in, but you can store an object in `EQ` with:

- `← PLOT ← EQ`  
- `← PLOT NXT 3D ← EQ`

**Affected by Flags:** None

**Related Commands:** RCEQ

---

**STIME**

**Serial Time-Out Command:** Specifies the period that SRECV (serial reception) and XMIT (serial transmission) wait before timing out.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x_{seconds}</code></td>
<td>→</td>
<td></td>
</tr>
<tr>
<td><code>0</code></td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** `← I/O NXT`  

**Affected by Flags:** None

**Remarks:** The value for `x` is interpreted as a positive value from 0 to 25.4 seconds. If no value is given, the default is 10 seconds. If `x` is 0,
there is no time-out; that is, the device waits indefinitely, which can drain the batteries.

STIME is not used for Kermit time-out.

**Related Commands:** BUFLEN, CLOSEIO, SBRK, SRECV, XMIT

---

**STO**

**Store Command:** Stores an object into a specified variable or object.

---

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>grob</td>
<td>PICT</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>obj</td>
<td>:n_port 'name_backup'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>obj</td>
<td>'name(index)'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>backup</td>
<td>n_port</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>library</td>
<td>n_port</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>library</td>
<td>:n_port :n_library</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** STO

**Affected by Flags:** None

**Remarks:** Storing a graphics object into PICT makes it the current graphics object.

To create a backup object, store the obj into the desired backup location (identified as :n_port 'name_backup'). STO will not overwrite an existing backup object.

To store backup objects and library objects, specify a port number (0 through 33). Ports 1 and 2 must be configured as independent RAM, since backup and library objects can be stored only in independent RAM (see the entry for FREE).

To use a library object, the object must be in a port and it must be attached. A library object from an application card (ROM) is
STO

automatically in a port (1 through 33), but a library object copied into RAM (such as through the PC Link) must be stored into a port using STO.

After storing a library object in a port, it must then be attached to its directory before it can be used. To make a stored library “attachable”, turn the calculator off and then on. (See the entry for ATTACH.) This action (storing a library object, then turning the calculator off and on) also causes the calculator to perform a system halt, which clears the stack, the LAST stack, and all local variables, and returns the MATH menu to the display.

STO can also replace a single element of an array or list stored in a variable. Specify the variable in level 1 as ‘name(index)’, which is a user function with index as the argument. The index can be n or n,m, where n specifies the row position in a vector or list, and n,m specifies the row-and-column position in a matrix.

Example: ‘A+B+C+D’ ‘SUMAD’ STO stores the expression A+B+C+D in the variable SUMAD.

5 ‘A(3)’ STO stores the integer 5 in the third element in a list or vector A.

2 ‘A(3,5)’ STO stores the integer 2 in the element in the third row and fifth column of matrix A.

Related Commands: DEFINE, RCL, →

STOALARM

Store Alarm Command: Stores an alarm in the system alarm list and returns its alarm index number.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{time}$</td>
<td>→</td>
<td>$n_{index}$</td>
</tr>
<tr>
<td>{ date time }</td>
<td>→</td>
<td>$n_{index}$</td>
</tr>
<tr>
<td>{ date time obj$^\text{action}$ }</td>
<td>→</td>
<td>$n_{index}$</td>
</tr>
<tr>
<td>{ date time obj$^\text{action}$ $x_{repeat}$ }</td>
<td>→</td>
<td>$n_{index}$</td>
</tr>
</tbody>
</table>
STOF

Keyboard Access: (TIME) ALRM STOA

Affected by Flags: Date Format (—42), Repeat Alarms Not Rescheduled (—43), Acknowledged Alarms Saved (—44)

Remarks: If the argument is a real number \(x_{\text{time}}\), the alarm date will be the current system date by default.

If \(obj_{\text{action}}\) is a string, the alarm is an appointment alarm, and the string is the alarm message. If \(obj_{\text{action}}\) is any other object type, the alarm is a control alarm, and the object is executed when the alarm comes due.

\(n_{\text{repeat}}\) is the repeat interval for the alarm in clock ticks, where 8192 clock ticks equals 1 second.

\(n_{\text{index}}\) is a real integer identifying the alarm based on its chronological position in the system alarm list.

Example: With flag —42 clear, this command:

\[
\{ 11.06 \ 15.2530 \ \text{RUN} \ 491520 \} \ \text{STOALARM}
\]

sets a repeating control alarm for November 6 of the currently specified year, at 3:25:30 PM. The alarm action is to execute variable \(\text{RUN}\). The repeat interval is 491520 clock ticks (1 minute).

Related Commands: DELALARM, FINDALARM, RCLALARM

---

STOF

Store Flags Command: Sets the states of the system flags or the system and user flags.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n_{\text{system}}</td>
<td>→</td>
<td>#n_{\text{system}}</td>
</tr>
<tr>
<td>{ #n_{\text{system}} \ #n_{\text{user}} }</td>
<td>→</td>
<td>{ #n_{\text{system}} \ #n_{\text{user}} }</td>
</tr>
</tbody>
</table>

Keyboard Access: (MODES) FLAG NXT STOF

Affected by Flags: Binary Integer Wordsize (—5 through —10)
**STOF**

The current wordsize must be 64 bits (the default wordsize) to store all flags. For example, executing STOF with a 32-bit binary integer stores only flags $-1$ through $-32$ and clears the other system flags.

**Remarks:** With argument $\#n_{\text{system}}$, STOF sets the states of the system flags ($-1$ through $-64$) only. With argument $\langle \#n_{\text{system}} \#n_{\text{user}} \rangle$, STOF sets the states of both the system and user flags.

A bit with value 1 sets the corresponding flag; a bit with value 0 clears the corresponding flag. The rightmost (least significant) bit of $\#n_{\text{system}}$ and $\#n_{\text{user}}$ correspond to the states of system flag $-1$ and user flag $+1$, respectively. If $\#n_{\text{system}}$ or $\#n_{\text{user}}$ contains fewer than 64 bits, the unspecified most significant bits are taken to have value 0.

STOF can preserve the states of flags before a program executes and changes the states. RCLF can then recall the flag’s states after the program is executed.

**Related Commands:** RCLF

---

**STOKEYS**

**Store Key Assignments Command:** Defines multiple keys on the user keyboard by assigning objects to specified keys.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ obj _x_key_1 \ldots _x_key_n }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ S _obj _x_key_1 \ldots _x_key_n }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>'S'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \([\text{MODES}] \text{ KEYS} \text{ STOK}\)

**Affected by Flags:** User-Mode Lock ($-61$) and User Mode ($-62$) affect the status of the user keyboard.

**Remarks:** $x_{key}$ is a real number of the form $rc.p$ specifying the key by its row number $r$, its column number $c$, and its plane (shift) $p$. (For a definition of plane, see the entry for ASN.)
The optional initial list parameter or argument S restores all keys without user assignments to their standard key assignments on the user keyboard. This is meaningful only when all standard key assignments had been suppressed (for the user keyboard) by the command 'S' DELKEYS (see DELKEYS).

If the argument obj is the name 'SKEY', the specified key is restored to its standard key assignment.

Related Commands: ASN, DELKEYS, RCLKEYS

---

STO+

Store Plus Command: Adds a number or other object to the contents of a specified variable.

\[
\begin{array}{ccc}
\text{Level 2} & \text{Level 1} & \rightarrow \\
\text{obj} & 'name' & \rightarrow \\
'name' & \text{obj} & \rightarrow \\
\end{array}
\]

Keyboard Access: \(\leftarrow\) [MEMORY] ARITH \ STO+

Affected by Flags: None

Remarks: The object on the stack and the object in the variable must be suitable for addition to each other. STO+ can add any combination of objects suitable for stack addition (see the entry for +).

Using STO+ to add two arrays (where obj is an array and name is the global name of an array) requires less memory than using the stack to add them.

Related Commands: STO-, STO*, STO/, +
STO–

**Store Minus Command:** Calculates the difference between a number (or other object) and the contents of a specified variable, and stores the new value to the specified variable.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>'name'</td>
<td>obj</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** («)(MEMORY) ARITH STO–

**Affected by Flags:** None

**Remarks:** The object on the stack and the object in the variable must be suitable for subtraction with each other. STO– can subtract any combination of objects suitable for stack subtraction (see the entry for −).

Using STO– to subtract two arrays (where obj is an array and name is the global name of an array) requires less memory than using the stack to subtract them.

**Related Commands:** STO+, STO*, STO/, −

STO*

**Store Times Command:** Multiplies the contents of a specified variable by a number or other object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>'name'</td>
<td>obj</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>
STO/

Keyboard Access: \(\rightarrow\) MEMORY ARITH STO\*.

Affected by Flags: None

Remarks: The object on the stack and the object in the variable must be suitable for multiplication with each other. When multiplying two arrays, the result depends on the order of the arguments. The new object of the named variable is the level 2 array times the level 1 array. The arrays must be conformable for multiplication.

Using STO\* to multiply two arrays or to multiply a number and an array (where \(obj\) is an array or a number and \(name\) is the global name of an array) requires less memory than using the stack to multiply them.

Related Commands: STO+, STO−, STO/, *

STO/

Store Divide Command: Calculates the quotient of a number (or other object) and the contents of a specified variable, and stores the new value to the specified variable.

\[
\begin{array}{ccc}
\text{Level 2} & \text{Level 1} & \rightarrow \\
\hline
\text{obj} & 'name' & \rightarrow \\
'name' & \text{obj} & \rightarrow \\
\end{array}
\]

Keyboard Access: \(\rightarrow\) MEMORY ARITH STO/.

Affected by Flags: None

Remarks: The new object of the specified variable is the level 2 object divided by the level 1 object.

The object on the stack and the object in the variable must be suitable for division with each other. If both objects are arrays, the divisor (level 1) must be a square matrix, and the dividend (level 2) must have the same number of columns as the divisor.
STO/

Using STO/ to divide one array by another array or to divide an array by a number (where \textit{obj} is an array or a number and \textit{name} is the global name of an array) requires less memory than using the stack to divide them.

\textbf{Related Commands:} \textit{STO+}, \textit{STO−}, \textit{STO∗}, \textit{/}

\begin{center}
\textbf{STOΣ}
\end{center}

\textbf{Store Sigma Command:} Stores \textit{obj} in the reserved variable $\Sigma DAT$.

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Level 1} & \rightarrow & \textbf{Level 1} \\
\hline
\textit{obj} & \rightarrow & \\
\hline
\end{tabular}
\end{center}

\textbf{Keyboard Access:} This command must be typed in, but you can store an object in $\Sigma DAT$ with either of the following:

\begin{itemize}
  \item \texttt{STAT} \texttt{ΣDAT}
  \item \texttt{ΣDAT}
\end{itemize}

\textbf{Affected by Flags:} None

\textbf{Remarks:} \textit{STOΣ} accepts any object and stores it in $\Sigma DAT$. However, if the object is not a matrix or the name of a variable containing a matrix, an \textit{Invalid Σ Data} error occurs upon subsequent execution of a statistics command.

\textbf{Related Commands:} \textit{CLΣ}, \textit{RCLΣ}, \textit{Σ+}, \textit{Σ−}
STREAM

Stream Execution Command: Moves the first two elements from the list onto the stack, and executes \( obj \). Then moves the next element (if any) onto the stack, and executes \( obj \) again using the previous result and the new element. Repeats this until the list is exhausted, and returns the final result.

\[
\begin{array}{ccc}
\text{Level 2} & \text{Level 1} & \rightarrow & \text{Level 1} \\
\{ \text{list} \} & \text{obj} & \rightarrow & \text{result}
\end{array}
\]

Keyboard Access: \[ \text{PRG LIST PROC STR} \]

Affected by Flags: None

Remarks: STREAM is nominally designed for \( obj \) to be a program or command that requires two arguments and returns one result.

Examples: \( (1 \ 2 \ 3 \ 4 \ 5) \leftrightarrow \) STREAM returns 120.
\( \leftrightarrow \) STREAM is equivalent to \( \Sigma \text{LIST} \).

Related Commands: DOSUBS

---

STR→

Evaluate String Command: Evaluates the text of a string as if the text were entered from the command line.

\[
\begin{array}{ccc}
\text{Level 1} & \rightarrow & \text{Level 1} \\
"\text{obj}" & \rightarrow & \text{evaluated-object}
\end{array}
\]

Keyboard Access: None. Must be typed in.

Affected by Flags: None
STR→

Remarks: OBJ→ also includes this function. STR→ is included for compatibility with the HP 28S.

Related Commands: ARRY→, DTAG, EQ→, LIST→, OBJ→, →STR

→STR

Object to String Command: Converts any object to string form.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>→</td>
<td>&quot;obj&quot;</td>
</tr>
</tbody>
</table>

Keyboard Access:

CHARS NXT →STR

PRG TYPE →STR

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12), Number Display Format (−49, −50)

Remarks: The full-precision internal form of a number is not necessarily represented in the result string. To ensure that →STR preserves the full precision of a number, select Standard number display format or a wordsize of 64 bits (or both) before executing →STR.

The result string includes the entire object, even if the displayed form of the object is too large to fit in the display.

If the argument object is normally displayed in two or more lines, the result string will contain newline characters (character 10) at the end of each line. The newlines are displayed as the character =.

If the argument object is already a string, →STR returns the string.

Example: →STR can create special displays to label program output or provide prompts for input. The sequence

"Result = " SWAP →STR + 1 DISP 1 FREEZE

3-334 Command Reference
displays \texttt{Result} = \texttt{object} in line 1 of the display, where \texttt{object} is a string form of an object taken from level 1.

\textbf{Related Commands:} \texttt{→ARRY, →LIST, STR→, →TAG, →UNIT}

---

### STWS

**Set Wordsize Command:** Sets the current binary integer wordsize to \( n \) bits, where \( n \) is a value from 1 through 64 (the default is 64).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>#n</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH} \) \text{BASE} \( \text{NXT} \) \text{STWS}

**Affected by Flags:** Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

**Remarks:** Values of \( n \) less than 1 or greater than 64 are interpreted as 1 or 64, respectively.

If the wordsize is smaller than an integer entered in the command line, then the most significant bits are not displayed upon entry. The truncated bits are still present internally (unless they exceed 64), but they are not used for calculations and they are lost when a command uses this binary integer as an argument.

Results that exceed the given wordsize are also truncated to the wordsize.

**Related Commands:** BIN, DEC, HEX, OCT, RCWS
Subset Command: Returns the portion of a string or list defined by specified positions, or returns the rectangular portion of a graphics object or PICT defined by two corner pixel coordinates.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>([[ matrix ]])_1</td>
<td>( n_{\text{startposition}} )</td>
<td>( n_{\text{endposition}} )</td>
<td>→</td>
<td>([[ matrix ]])_2</td>
</tr>
<tr>
<td>([[ matrix ]])_1</td>
<td>{ ( n_{\text{row}} ) ( n_{\text{column}} ) }</td>
<td>( n_{\text{endposition}} )</td>
<td>→</td>
<td>([[ matrix ]])_2</td>
</tr>
<tr>
<td>([[ matrix ]])_1</td>
<td>( n_{\text{startposition}} )</td>
<td>{ ( n_{\text{row}} ) ( n_{\text{column}} ) }</td>
<td>→</td>
<td>([[ matrix ]])_2</td>
</tr>
<tr>
<td>([[ matrix ]])_1</td>
<td>{ ( n_{\text{row}} ) ( n_{\text{column}} ) }</td>
<td>( n_{\text{endposition}} )</td>
<td>→</td>
<td>([[ matrix ]])_2</td>
</tr>
</tbody>
</table>

"string\(_{\text{target}}" \( n_{\text{startposition}} \) | \( n_{\text{endposition}} \) | → "string\(_{\text{result}}" \}

\( \{ \text{list\(_{\text{target}}\) \} \) \( n_{\text{startposition}} \) | \( n_{\text{endposition}} \) | → \( \{ \text{list\(_{\text{result}}\) \} \)

\( \text{grob\(_{\text{target}}\) \} \) \( \#n_{1} \#m_{1} \) | \( \#n_{2} \#m_{2} \) | → \( \text{grob\(_{\text{result}}\) \} \)

\( \text{PICT} \) \( (x_{1}, y_{1}) \) | \( (x_{2}, y_{2}) \) | → \( \text{grob\(_{\text{result}}\) \} \)

Keyboard Access:

\( \leftarrow \) CHARS SUB

PRG LIST SUB

PRG GROB SUB

MTH MATR MAKE NXT SUB

Affected by Flags: None

Remarks: If \( n_{\text{end position}} \) is less than \( n_{\text{start position}} \), SUB returns an empty string or list. Values of \( n \) less than 1 are treated as 1; values of \( n \) exceeding the length of the string or list are treated as that length.

For graphics objects, a user-unit coordinate less than the minimum user-unit coordinate of the graphics object is treated as that minimum. A pixel or user-unit coordinate greater than the maximum pixel or user-unit coordinate of the graphics object is treated as that maximum.

Examples: \( \{ A B C D E \} \) \( 2 4 \) SUB returns \( \{ B C D \} \).
SVD

Singular Value Decomposition Command: Returns the singular value decomposition of an \( m \times n \) matrix.

\[
\begin{align*}
&\text{Level 1} \quad \rightarrow \quad \text{Level 3} \quad \text{Level 2} \quad \text{Level 1} \\
\begin{bmatrix} \text{matrix} \end{bmatrix}_A \quad \rightarrow \quad \begin{bmatrix} \text{matrix} \end{bmatrix}_U \quad \begin{bmatrix} \text{matrix} \end{bmatrix}_V \quad \begin{bmatrix} \text{vector} \end{bmatrix}_S
\end{align*}
\]

Keyboard Access: \( \text{MTH} \quad \text{MATR} \quad \text{FACTR} \quad \text{SVD} \)

Affected by Flags: None

Remarks: SVD decomposes \( A \) into 2 matrices and a vector. \( U \) is an \( m \times m \) orthogonal matrix, \( V \) is an \( n \times n \) orthogonal matrix, and \( S \) is a real vector, such that \( A = U \times \text{diag}(S) \times V \). \( S \) has length \( \text{MIN}(m,n) \) and contains the singular values of \( A \) in nonincreasing order. The matrix \( \text{diag}(S) \) is an \( m \times n \) diagonal matrix containing the singular values \( S \).

The computed results should minimize (within computational precision):

\[
\frac{|A - U \cdot \text{diag}(S) \cdot V|}{\text{min}(m,n) \cdot |A|}
\]

where \( \text{diag}(S) \) denotes the \( m \times n \) diagonal matrix containing the singular values \( S \).

Related Commands: DIAG→, MIN, SVL
SVL

**Singular Values Command:** Returns the singular values of an $m \times n$ matrix.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[[\text{matrix}]]$</td>
<td>→</td>
<td>$[\text{vector}]$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH MATR FACTR NXT] SVL

**Affected by Flags:** None

**Remarks:** SVL returns a real vector that contains the singular values of an $m \times n$ matrix in nonincreasing order. The vector has length $\text{MIN}(m,n)$.

**Related Commands:** MIN, SVD

SWAP

**Swap Objects Command:** Interchanges the first two objects on the stack.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{obj}_1$</td>
<td>$\text{obj}_2$</td>
<td>→</td>
<td>$\text{obj}_2$</td>
<td>$\text{obj}_1$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [SWAP]

**Affected by Flags:** None

**Related Commands:** DUP, DUPN, DUP2, OVER, PICK, ROLL, ROLLD, ROT
SYSEVAL

Evaluate System Object Command: Evaluates unnamed operating system objects specified by their memory addresses.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n_ADDRESS</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Using SYSEVAL with random addresses can corrupt memory.

Example: Display the version letter of an HP 48 by executing #30794H SYSEVAL. Version A, for example, would display "HPHP48-A".

Related Commands: EVAL, LIBEVAL

Percent of Total Function: Returns the percent of the level 2 argument that is represented by the level 1 argument.
%T

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>→</td>
<td>100y/x</td>
</tr>
<tr>
<td>x</td>
<td>'symb'</td>
<td>→</td>
<td>'%T(x, symb)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>x</td>
<td>→</td>
<td>'%T(symb, x)'</td>
</tr>
<tr>
<td>'symb1'</td>
<td>'symb2'</td>
<td>→</td>
<td>'%T(symb1, symb2)'</td>
</tr>
<tr>
<td>x_unit1</td>
<td>y_unit2</td>
<td>→</td>
<td>100y_unit2 / x_unit1</td>
</tr>
<tr>
<td>x_unit</td>
<td>'symb'</td>
<td>→</td>
<td>'%T(x_unit, symb)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>x_unit</td>
<td>→</td>
<td>'%T(symb, x_unit)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [REAL] [%T]

**Affected by Flags:** Numerical Results (−3)

**Remarks:** If both arguments are unit objects, the units must be consistent with each other.

The dimensions of a unit object are dropped from the result, but units are part of the calculation.

For more information on using temperature units with arithmetic functions, refer to the entry for +.

**Example:** 1_m 500_cm %T returns 500, because 500 cm represents 500% of 1 m.

100_K 50_K %T returns 50.

**Related Commands:** %, %CH
TAIL

---TAG

**Stack to Tag Command:** Combines objects in levels 1 and 2 to create tagged (labeled) object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>&quot;tag&quot;</td>
<td>→</td>
<td>:tag:obj</td>
</tr>
<tr>
<td>obj</td>
<td>'name'</td>
<td>→</td>
<td>:name:obj</td>
</tr>
<tr>
<td>obj</td>
<td>x</td>
<td>→</td>
<td>:x:obj</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] TYPE →TAG

**Affected by Flags:** None

**Remarks:** The "tag" argument is a string of fewer than 256 characters.

**Related Commands:** →ARRY, DTAG, →LIST, OBJ→, →STR, →UNIT

TAIL

**Last Listed Elements Command:** Returns all but the first element of a list or string.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ obj₁ ... objₙ } → { obj₂ ... objₙ }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;string₁&quot; → &quot;string₂&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:**

[PRG] LIST ELEM NXT TAIL

[CHARS] NXT TAIL
TAIL

Affected by Flags: None

Example: "\texttt{\textbackslash t \textbackslash a \textbackslash l \textbackslash l}" TAIL returns "\texttt{\textbackslash a \textbackslash l \textbackslash l}".

Related Commands: HEAD

---

TAN

Tangent Analytic Function: Returns the tangent of the argument.

\[
\begin{array}{|c|c|}
\hline
\text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
z & \rightarrow & \tan z \\
'symb' & \rightarrow & '\text{TAN}(symb)' \\
x_\text{unit}\text{\_angular} & \rightarrow & \tan (x_\text{unit}\text{\_angular}) \\
\hline
\end{array}
\]

Keyboard Access: \textbf{TAN}

Affected by Flags: Numerical Results (–3), Angle Mode (–17, –18), Infinite Result Exception (–22)

Remarks: For real arguments, the current angle mode determines the number’s interpretation as an angle, unless the angular units are specified.

For a real argument that is an odd-integer multiple of 90 in Degrees mode, an \texttt{Infinite Result} exception occurs. If flag –22 is set (no error), the sign of the result (MAXR) matches that of the argument.

For complex arguments,

\[
\tan(x + iy) = \frac{(\sin x)(\cos x) + i(\sinh y)(\cosh y)}{\sinh^2 y + \cos^2 x}
\]

If the argument for TAN is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing TAN with a unit object, the angle mode must be set to Radians (since this is a “neutral” mode).
Related Commands: ATAN, COS, SIN

\[ \text{TANH} \]

**Hyperbolic Tangent Analytic Function:** Returns the hyperbolic tangent of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z )</td>
<td>→</td>
<td>( \tanh z )</td>
</tr>
<tr>
<td>( 'symb' )</td>
<td>→</td>
<td>( 'TANH(symb)' )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [HYP] [TANH]

**Affected by Flags:** Numerical Results (-3)

**Remarks:** For complex arguments,

\[
\tanh(x + iy) = \frac{\sinh 2x + i \sin 2y}{\cosh 2x + \cos 2y}
\]

**Related Commands:** ATANH, COSH, SINH

\[ \text{TAYLR} \]

**Taylor’s Polynomial Command:** Calculates the \( n \)th order Taylor’s polynomial of \( 'symb' \) in the variable global.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 'symb' )</td>
<td>( 'global' )</td>
<td>( n_{\text{order}} )</td>
<td>→</td>
<td>( 'symb_{\text{Taylor}}' )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [SYM] [SYMBOLIC] [TAYLR]
**TAYLR**

**Affected by Flags:** None

**Remarks:** The polynomial is calculated at the point \( global = 0 \) (called a MacLaurin series).

TAYLR always returns a symbolic result, regardless of the state of the Numeric Results flag \((-3)\).

**Example:** The command sequence '1+\sin(x)^2' 'x' 5 TAYLR returns '1+x^2-8/4!x^4'.

**Related Commands:** \( \partial, \int, \Sigma \)

---

**TDELTA**

**Temperature Delta Function:** Calculates a temperature change.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y )</td>
<td>→</td>
<td>( x_{\Delta} )</td>
</tr>
<tr>
<td>( x_{\text{unit}1} )</td>
<td>( y_{\text{unit}2} )</td>
<td>→</td>
<td>( x_{\text{unit}1_{\Delta}} )</td>
</tr>
<tr>
<td>( x_{\text{unit}} )</td>
<td>( 'symb' )</td>
<td>→</td>
<td>'TDELTA(x_{\text{unit}},symb)'</td>
</tr>
<tr>
<td>( 'symb' )</td>
<td>( y_{\text{unit}} )</td>
<td>→</td>
<td>'TDELTA(symb,y_{\text{unit}})'</td>
</tr>
<tr>
<td>( 'symb_1' )</td>
<td>( 'symb_2' )</td>
<td>→</td>
<td>'TDELTA(symb_1,symb_2)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (EQ LIB) UTILS NXT TDELTA

**Affected by Flags:** Numerical Results \((-3)\)

**Remarks:** TDELTA subtracts two points on a temperature scale, yielding a temperature increment (not an actual temperature). \( x \) or \( x_{\text{unit}} \) is the final temperature, and \( y \) or \( y_{\text{unit}} \) is the initial temperature. If unit objects are given, the increment is returned as a unit object with the same units as \( x_{\text{unit}} \). If real numbers are given, the increment is returned as a real number.

**Related Commands:** TINC

3-344 Command Reference
TEACH

Teaching Examples Function: Creates an EXAMPLES (EXAM) subdirectory in the HOME directory and loads HP 48 programming, graphing, and solver examples from ROM into it.

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Items stored in the EXAMPLES subdirectory are deleted when CLTEACH is executed.

Related Commands: CLTEACH

TEXT

Show Stack Display Command: Displays the stack display.

Keyboard Access: [PRG] [NXT] [OUT] TEXT

Affected by Flags: None

Remarks: TEXT switches from the graphics display to the stack display. TEXT does not update the stack display.

Example: The command sequence DRAW 5 WAIT TEXT selects the graphics display and plots the contents of the reserved variable EQ (or reserved variable ΣDAT). It subsequently waits for 5 seconds, and then switches back from the graphics display to the stack display.

Related Commands: PICTURE, PVIEW
THEN

THEN Command: Starts the true-clause in conditional or error-trapping structure.

See the IF and IFERR entries for syntax information.

Keyboard Access:

```
PRG NXT ERROR IFERR THEN
PRG BRCH CASE THEN
PRG BRCH IF THEN
```

Remarks: See the IF and IFERR entries for more information.

Related Commands: CASE, ELSE, END, IF, IFERR

---

TICKS

Ticks Command: Returns the system time as a binary integer, in units of $1/8192$ second.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>( n_{\text{time}} )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \arrowleft \) TIME Ticks

Affected by Flags: None

Remarks: TICKS enables elapsed time computations.

Example: If the result from a previous invocation from TICKS is in level 1, then `TICKS SWAP - B>R 8192 /` returns a real number whose value is the elapsed time in seconds between the two invocations.

Related Commands: TIME
**TIME**

**Time Command:** Returns the system time in the form HH.MMSSs.

Keyboard Access:  

Affected by Flags: None

Remarks: time has the form HH.MMSSs, where HH is hours, MM is minutes, SS is seconds, and s is zero or more digits (as many as allowed by the current display mode) representing fractional seconds. time is always returned in 24-hour format, regardless of the state of the Clock Format flag (-41).

Related Commands: DATE, TICKS, TSTR

---

**→TIME**

**Set System Time Command:** Sets the system time.

Keyboard Access:  

Affected by Flags: None

Remarks: time must have the form HH.MMSSs, where HH is hours, MM is minutes, SS is seconds, and s is zero or more digits (as many as allowed by the current display mode) representing fractional seconds. time must use 24-hour format.
**TIME**

**Example:** 13:33:41 \( \rightarrow \) TIME sets the system time to 1:33:41 PM.

**Related Commands:** CLKADJ, \( \rightarrow \) DATE

---

**TINC**

**Temperature Increment Command:** Calculates a temperature increment.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{\text{initial}} )</td>
<td>( y_{\Delta \text{t}} )</td>
<td>( \rightarrow )</td>
<td>( x_{\text{final}} )</td>
</tr>
<tr>
<td>( x_{\text{unit1}} )</td>
<td>( y_{\text{unit2}_{\Delta \text{t}}} )</td>
<td>( \rightarrow )</td>
<td>( x_{\text{unit1}_{\text{final}}} )</td>
</tr>
<tr>
<td>( x_{\text{unit}} )</td>
<td>'symb'</td>
<td>( \rightarrow )</td>
<td>'TINC(( x_{\text{unit}},\text{symb} ) )'</td>
</tr>
<tr>
<td>'symb'</td>
<td>( y_{\text{unit}_{\Delta \text{t}}} )</td>
<td>( \rightarrow )</td>
<td>'TINC(symb,( y_{\text{unit}_{\Delta \text{t}}} ) )'</td>
</tr>
<tr>
<td>'symb1'</td>
<td>'symb2'</td>
<td>( \rightarrow )</td>
<td>'TINC(symb1,symb2)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \leftrightarrow \) (EQ LIB) UTILS \( \rightarrow \) TINC

**Affected by Flags:** Numerical Results \( \rightarrow \) (–3)

**Remarks:** TINC adds a temperature *increment* (not an actual temperature) to a point on a temperature scale. Use a negative increment to subtract the increment from the temperature. \( x_{\text{initial}} \) or \( x_{\text{unit1}} \) is the initial temperature, and \( y_{\Delta \text{t}} \) or \( y_{\text{unit2}_{\Delta \text{t}}} \) is the temperature increment. The returned temperature is the resulting final temperature. If unit objects are given, the final temperature is returned as a unit object with the same units as \( x_{\text{unit1}} \). If real numbers are given, the final temperature is returned as a real number.

**Related Commands:** TDELTA
**TLINE**

**Toggle Line Command:** For each pixel along the line in *PICT* defined by the specified coordinates, TLINE turns off every pixel that is on, and turns on every pixel that is off.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x₁, y₁)</td>
<td>(x₂, y₂)</td>
<td>→</td>
<td>{ #n₁ #m₁ }</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] [PICT] TLINE

**Affected by Flags:** None

**Example:** The following program toggles on and off 10 times the pixels on the line defined by user-unit coordinates (1,1) and (9,9). Each state is maintained for .25 seconds.

```
<
ERASE 0 10 XRNG 0 10 YNFG
( # 0d # 0d ) PVIEC
<
1 10 START
(1,1) (9,9) TLINE
.25 WAIT
NEXT
>
```

**Related Commands:** ARC, BOX, LINE
TMENU

Temporary Menu Command: Displays a built-in menu, library menu, or user-defined menu.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_{menu}</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>{ list_{definition} }</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>'name_{definition}'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: ←MODES MENU TMENU

Affected by Flags: None

Remarks: TMENU works just like MENU, except for user-defined menus (specified by a list or by the name of a variable that contains a list). Such menus are displayed like a custom menu and work like a custom menu, but are not stored in reserved variable CST. Thus, a menu defined and displayed by TMENU cannot be redisplayed by evaluating CST.

See the MENU entry for a list of the HP 48 built-in menus and the corresponding menu numbers (x_{menu}).

Examples: 7 TMENU displays the first page of the MTH MATR menu.
48.02 TMENU displays the second page of the UNITS MASS menu.
768 TMENU displays the first page of commands in library 768.
\{ A 123 "ABC" \} TMENU displays the custom menu defined by the list argument.
'MYMENU' TMENU displays the custom menu defined by the name argument.

Related Commands: MENU, RCLMENU
**TOT**

**Total Command:** Computes the sum of each of the $m$ columns of coordinate values in the current statistics matrix (reserved variable $\Sigma DAT$).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>$x_{sum}$</td>
</tr>
<tr>
<td></td>
<td>→</td>
<td>$[x_{sum1} \ x_{sum2} \ \cdots \ x_{summ}]$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** $\leftarrow$(STAT) 1VAR  TOT

**Affected by Flags:** None

**Remarks:** The sums are returned as a vector of $m$ real numbers, or as a single real number if $m = 1$.

**Related Commands:** MAXΣ, MINΣ, MEAN, PSDEV, PVAR, SDEV, VAR

---

**TRACE**

**Matrix Trace Command:** Returns the trace of a square matrix.

$$\{ \}$$

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[matrix]]_{n \times n}</td>
<td>→</td>
<td>$x_{trace}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH MATR HORN NXT TRACE

**Affected by Flags:** None

**Remarks:** The trace of a square matrix is the sum of its diagonal elements.
**TRANSIO**

I/O Translation Command: Specifies the character translation option. These translations affect only ASCII Kermit transfers and files printed to the serial port.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{\text{option}} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( \leftarrow \text{I/O} \) IOPAR TRAN

Affected by Flags: None

Remarks: Legal values for \( n \) are as follows:

<table>
<thead>
<tr>
<th>( n )</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No translation</td>
</tr>
<tr>
<td>1</td>
<td>Translate character 10 (line feed only) to/from characters 10 and 13 (line feed with carriage return, the Kermit protocol) (the default value)</td>
</tr>
<tr>
<td>2</td>
<td>Translate characters 128 through 159 (80 through 9F hexadecimal)</td>
</tr>
<tr>
<td>3</td>
<td>Translate all characters (128 through 255)</td>
</tr>
</tbody>
</table>

Related Commands: BAUD, CKSM, PARITY
TRNC

TRN

Transpose Matrix Command: Returns the (conjugate) transpose of a matrix.

Keyboard Access: MTH MATR MAKE TRN

Affected by Flags: None

Remarks: TRN replaces an $n \times m$ matrix $A$ with an $m \times n$ matrix $A^T$, where:

- $A_{ij}^T = A_{ji}$ for real matrices
- $A_{ij}^T = \text{CONJ}(A_{ji})$ for complex matrices

If the matrix is specified by name, $A^T$ replaces $A$ in name.

Example: $[[2 \ 3 \ 1] [4 \ 6 \ 9]]$ TRN returns $[[2 \ 4] [3 \ 6] [1 \ 9]]$.

Related Commands: CONJ

TRNC

Truncate Function: Truncates an object to a specified number of decimal places or significant digits, or to fit the current display format.
**TRNC**

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z_1 )</td>
<td>( n_{\text{truncate}} )</td>
<td>( \rightarrow )</td>
<td>( z_2 )</td>
</tr>
<tr>
<td>( z_1 )</td>
<td>( 'symb_{\text{truncate}}' )</td>
<td>( \rightarrow )</td>
<td>( '\text{TRNC}(z_1,\text{symb}_{\text{truncate}})' )</td>
</tr>
<tr>
<td>( 'symb_1' )</td>
<td>( n_{\text{truncate}} )</td>
<td>( \rightarrow )</td>
<td>( '\text{TRNC}(\text{symb}<em>1,n</em>{\text{truncate}})' )</td>
</tr>
<tr>
<td>( 'symb_1' )</td>
<td>( 'symb_{\text{truncate}}' )</td>
<td>( \rightarrow )</td>
<td>( '\text{TRNC}(\text{symb}<em>1,\text{symb}</em>{\text{truncate}})' )</td>
</tr>
<tr>
<td>( [\text{array}]_1 )</td>
<td>( n_{\text{truncate}} )</td>
<td>( \rightarrow )</td>
<td>( [\text{array}]_2 )</td>
</tr>
<tr>
<td>( x_{_\text{unit}} )</td>
<td>( n_{\text{truncate}} )</td>
<td>( \rightarrow )</td>
<td>( y_{_\text{unit}} )</td>
</tr>
<tr>
<td>( x_{_\text{unit}} )</td>
<td>( 'symb_{\text{truncate}}' )</td>
<td>( \rightarrow )</td>
<td>( '\text{TRNC}(x_{_\text{unit}},\text{symb}_{\text{truncate}})' )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [MTH] [REAL] [NXT] [NXT] TRNC

**Affected by Flags:** Numerical Results (−3)

**Remarks:** \( n_{\text{truncate}} \) (or \( 'symb_{\text{truncate}}' \) if flag −3 is set) controls how the level 2 argument is truncated, as follows:

<table>
<thead>
<tr>
<th>( n_{\text{truncate}} )</th>
<th>Effect on Level 2 Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 through 11</td>
<td>truncated to ( n ) decimal places</td>
</tr>
<tr>
<td>−1 through −11</td>
<td>truncated to ( n ) significant digits</td>
</tr>
<tr>
<td>12</td>
<td>truncated to the current display format</td>
</tr>
</tbody>
</table>

For complex numbers and arrays, each real number element is truncated. For unit objects, the number part of the object is truncated.

**Examples:** (4.5792, 8.1275) 2 TRNC returns (4.57, 8.12).

\[ [2.34907 \ 3.96351 \ 2.73453] \ -2 \] TRNC returns \[ [2.3 \ 3.9 \ 2.7] \].

**Related Commands:** RND
TRUTH

Truth Plot Type Command: Sets the plot type to TRUTH.

Keyboard Access: Function key \( \text{PLOT} \) followed by \( \text{PTYPE} \) \( \text{TRUTH} \)

Affected by Flags: None

Remarks: When the plot type is TRUTH, the DRAW command plots the current equation as a truth-valued function of two real variables. The current equation is specified in the reserved variable \( EQ \). The plotting parameters are specified in the reserved variable \( PPAR \), which has this form:

\[
\langle (x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}}), \text{indep}, \text{res}, \text{axes}, \text{ptype}, \text{depend} \rangle
\]

For plot type TRUTH, the elements of \( PPAR \) are used as follows:

- \( (x_{\text{min}}, y_{\text{min}}) \) is a complex number specifying the lower left corner of \( PICT \) (the lower left corner of the display range). The default value is \(( -6.5, -3.1 )\).

- \( (x_{\text{max}}, y_{\text{max}}) \) is a complex number specifying the upper right corner of \( PICT \) (the upper right corner of the display range). The default value is \(( 6.5, 3.2 )\).

- \( \text{indep} \) is a name specifying the independent variable on the horizontal axis, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the horizontal plotting range). The default value is \( X \).

- \( \text{res} \) is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable on the horizontal axis, or a binary integer specifying that interval in pixels. The default value is \( \emptyset \), which specifies an interval of 1 pixel.

- \( \text{axes} \) is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is \( \langle \emptyset, \emptyset \rangle \).

- \( \text{ptype} \) is a command name specifying the plot type. Executing the command TRUTH places the name TRUTH in \( \text{ptype} \).

- \( \text{depend} \) is a name specifying the independent variable on the vertical axis, or a list containing such a name and two numbers specifying
TRUTH

the minimum and maximum values for the independent variable on the vertical axis (the vertical plotting range). The default value is \( Y \).

The contents of \( EQ \) must be an expression or program, and cannot be an equation. It is evaluated for each pixel in the plot region. The minimum and maximum values of the independent variables (the plotting ranges) can be specified in \( indep \) and \( depend \); otherwise, the values in \( (x_{\min}, y_{\min}) \) and \( (x_{\max}, y_{\max}) \) (the display range) are used. The result of each evaluation must be a real number. If the result is zero, the state of the pixel is unchanged. If the result is nonzero, the pixel is turned on (made dark).

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, WIREFRAME, YSLICE

TSTR

Date and Time String Command: Returns a string derived from the date and time.

```
{ }
```

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>time</td>
<td>→</td>
<td>&quot;DOW DATE TIME&quot;</td>
</tr>
</tbody>
</table>

Keyboard Access: \(<\text{TIME}>\text{NXT}\text{NXT}T\text{STR}>\)

Affected by Flags: Date Format (—42), Time Format (—41)

Remarks: The string has the form "\( DOW \) \( DATE \) \( TIME \)",
where \( DOW \) is a three-letter abbreviation of the day of the week corresponding to the argument \( date \) and \( time \), \( DATE \) is the argument \( date \) in the current date format, and \( TIME \) is the argument \( time \) in the current time format.

Example: With flags —42 and —41 clear, 2.061990 14.55 TSTR returns "TUE 02/06/90 02:55:00P".

3-356 Command Reference
Related Commands: DATE, TICKS, TIME

**TVARS**

**Typed Variables Command:** Lists all global variables in the current directory that contain objects of the specified types.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{type}$</td>
<td>→</td>
<td>{ global ... }</td>
</tr>
<tr>
<td>{ $n_{type}$ ... }</td>
<td>→</td>
<td>{ global ... }</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (MEMORY) [IFE TWH]

**Affected by Flags:** None

**Remarks:** If the current directory contains no variables of the specified types, TVARS returns an empty list.

For a table of the object-type numbers, see the entry for TYPE.

**Related Commands:** PVARS, TYPE, VARS

---

**TVM**

**TVM Menu Command:** Displays the TVM Solver menu.

**Keyboard Access:** This command must be typed in, but you can also access the menu using (SOLVE) [SOLVR].

**Affected by Flags:** None

**Related Commands:** AMORT, TVMBEG, TVMEND, TVMROOT
TVMBEG

Payment at Start of Period Command: Specifies that TVM calculations treat payments as being made at the beginning of the compounding periods.

Keyboard Access: This command must be typed in, but you can control begin/end mode with \( \text{SOLVE} \) TVM BEG.

Affected by Flags: None

Related Commands: AMORT, TVM, TVMEND, TVMROOT

TVMEND

Payment at End of Period Command: Specifies that TVM calculations treat payments as being made at the end of the compounding periods.

Keyboard Access: This command must be typed in, but you can control begin/end mode with \( \text{SOLVE} \) TVM BEG.

Affected by Flags: None

Related Commands: AMORT, TVM, TVMBEG, TVMROOT

TVMROOT

TVM Root Command: Solves for the specified TVM variable using values from the remaining TVM variables.

Keyboard Access: \( \text{SOLVE} \) TVM TVMR

3-358 Command Reference
**TYPE**

**Type Command:** Returns the type number of an object.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>obj</code></td>
<td>→</td>
<td><code>n_{\text{type}}</code></td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- `PRG TYPE NXT TYPE`  
- `PRG TEST NXT TYPE`  

**Affected by Flags:** None

**Remarks:** The following table lists object types and their type numbers.

**Object Type Numbers**

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Objects:</td>
<td></td>
</tr>
<tr>
<td>Real number</td>
<td>0</td>
</tr>
<tr>
<td>Complex number</td>
<td>1</td>
</tr>
<tr>
<td>Character string</td>
<td>2</td>
</tr>
<tr>
<td>Real array</td>
<td>3</td>
</tr>
<tr>
<td>Complex array</td>
<td>4</td>
</tr>
<tr>
<td>List</td>
<td>5</td>
</tr>
<tr>
<td>Global name</td>
<td>6</td>
</tr>
<tr>
<td>Local name</td>
<td>7</td>
</tr>
<tr>
<td>Program</td>
<td>8</td>
</tr>
<tr>
<td>Algebraic object</td>
<td>9</td>
</tr>
<tr>
<td>Binary integer</td>
<td>10</td>
</tr>
</tbody>
</table>
### Object Type Numbers (continued)

<table>
<thead>
<tr>
<th>Object Type Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics object</td>
<td>11</td>
</tr>
<tr>
<td>Tagged object</td>
<td>12</td>
</tr>
<tr>
<td>Unit object</td>
<td>13</td>
</tr>
<tr>
<td>XLIB name</td>
<td>14</td>
</tr>
<tr>
<td>Directory</td>
<td>15</td>
</tr>
<tr>
<td>Library</td>
<td>16</td>
</tr>
<tr>
<td>Backup object</td>
<td>17</td>
</tr>
</tbody>
</table>

**Built-in Commands:**
- Built-in function: 18
- Built-in command: 19

**System Objects:**
- System binary: 20
- Extended real: 21
- Extended complex: 22
- Linked array: 23
- Character: 24
- Code object: 25
- Library data: 26
- External object: 27-31

The HP 28S TYPE command returns number 8 for built-in functions and built-in commands (HP 48 TYPE numbers 18 and 19).

**Related Commands:** SAME, TVARS, VTYPE, ==

---

### UBASE

**Convert to SI Base Units Function:** Converts a unit object to SI base units.
**UFACT**

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{\text{unit}}$</td>
<td>→</td>
<td>$y_{\text{base-units}}$</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'UBASE(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (UNIT) UBASE

**Affected by Flags:** Numerical Results (-3)

**Example:** 30 knot UBASE returns 15.433333333 m/s.

**Related Commands:** CONVERT, UFACT, ¬UNIT, UVAL

---

**UFACT**

**Factor Unit Command:** Factors the level 1 unit from the unit expression of the level 2 unit object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1_{\text{unit}_1}$</td>
<td>$x_2_{\text{unit}_2}$</td>
<td>→</td>
<td>$x_3_{\text{unit}_2*\text{unit}_3}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (UNIT) UFACT

**Affected by Flags:** None

**Remarks:** UFACT is equivalent to this sequence:

```
OBJ→ 3 ROLLD / OVER / UBASE *
```

**Example:** 1 Wh 1 N UFACT returns 1 N*m/s.

**Related Commands:** CONVERT, UBASE, ¬UNIT, UVAL
**→UNIT**

**Stack to Unit Object Command:** Creates a unit object from a real number and the unit part of a unit object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y_unit</td>
<td>→</td>
<td>x_unit</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

- `{PRG}`
- `{TYPE}`
- `{UNIT}`
- `{UNITS}`

**Affected by Flags:** None

**Remarks:** →UNIT adds units to a real number, combining the number and the unit part of a unit object (the numerical part of the unit object is ignored). →UNIT is the reverse of OBJ→ applied to a unit object.

**Related Commands:** →ARRY, →LIST, →STR, →TAG

---

**UNTIL**

**UNTIL Command:** Starts test-clause in a DO ... UNTIL ... END indefinite loop structure.

See the DO entry for syntax information.

**Keyboard Access:** `{PRG}` `{BRCH}` `{DO}` `{UNTIL}`

**Remarks:** See the DO entry for more information.

**Related Commands** DO, END
UPDIR

**Up Directory Command:** Makes the parent of the current directory the new current directory.

**Keyboard Access:** \(<\text{UP}\>\)

**Affected by Flags:** None

**Remarks:** UPDIR has no effect if the current directory is HOME.

**Related Commands:** CRDIR, HOME, PATH, PGDIR

---

UTPC

**Upper Chi-Square Distribution Command:** Returns the probability utpc(n, x) that a chi-square random variable is greater than x, where n is the number of degrees of freedom of the distribution.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>x</td>
<td>→</td>
<td>utpc(n, x)</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\text{MTH NXT PROB NXT UTPC}\)

**Affected by Flags:** None

**Remarks:** The defining equations are these:

- For \(x \geq 0\):
  \[
  utpc(n, x) = \left[ \frac{1}{2^n \Gamma \left( \frac{n}{2} \right)} \right] \int_x^\infty t^{n/2 - 1} e^{-\frac{1}{2} t} dt
  \]

- For \(x < 0\):
  \[
  utpc(n, x) = 1
  \]

For any value \(z\), \(\Gamma \left( \frac{z}{2} \right) = \left( \frac{z}{2} - 1 \right)!, \) where ! is the HP 48 factorial command.
UTPC

The value \( n \) is rounded to the nearest integer and, when rounded, must be positive.

**Related Commands:** UTPF, UTPN, UTPT

---

**UTPF**

**Upper Snedecor's F Distribution Command:** Returns the probability \( \text{utpf}(n_1, n_2, x) \) that a Snedecor's F random variable is greater than \( x \), where \( n_1 \) and \( n_2 \) are the numerator and denominator degrees of freedom of the F distribution.

\[
\begin{array}{c|c|c|c|c}
\text{Level 3} & \text{Level 2} & \text{Level 1} & \rightarrow & \text{Level 1} \\
\hline
n_1 & n_2 & x & \rightarrow & \text{utpf}(n_1, n_2, x) \\
\end{array}
\]

**Keyboard Access:** MTH NXT PROB NXT UTPF

**Affected by Flags:** None

**Remarks:** The defining equations for \( \text{utpf}(n_1, n_2, x) \) are these:

- For \( x \geq 0 \):
  \[
  \left( \frac{n_1}{n_2} \right)^{\frac{n_1}{2}} \left[ \frac{\Gamma \left( \frac{n_1+n_2}{2} \right)}{\Gamma \left( \frac{n_1}{2} \right) \Gamma \left( \frac{n_2}{2} \right)} \right] \int_x^\infty t^{\frac{(n_1-2)}{2}} \left[ 1 + \left( \frac{n_1}{n_2} \right) t \right]^{\frac{(n_1+n_2)}{2}} dt 
  \]

- For \( x < 0 \):
  \[
  \text{utpf}(n_1, n_2, x) = 1
  \]

For any value \( z \), \( \Gamma \left( \frac{z}{2} \right) = \left( \frac{z}{2} - 1 \right)! \), where ! is the HP 48 factorial command.

The values \( n_1 \) and \( n_2 \) are rounded to the nearest integers and, when rounded, must be positive.

**Related Commands:** UTPC, UTPN, UTPT
**UTPN**

**Upper Normal Distribution Command:** Returns the probability \( \text{utpn}(m, v, x) \) that a normal random variable is greater than \( x \), where \( m \) and \( v \) are the mean and variance, respectively, of the normal distribution.

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m )</td>
<td>( v )</td>
<td>( x )</td>
<td>→</td>
<td>utpn(m, v, x)</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH NXT PROB NXT UTPT} \)

**Affected by Flags:** None

**Remarks:** For all \( x \) and \( m \), and for \( v > 0 \), the defining equation is this:

\[
\text{utpn}(m, v, x) = \left[ \frac{1}{\sqrt{2\pi v}} \right] \int_x^{\infty} e^{-\frac{(t-m)^2}{2v}} \, dt
\]

For \( v = 0 \), \( \text{UTPN} \) returns 0 for \( x \geq m \), and 1 for \( x < m \).

**Related Commands:** UTPC, UTPF, UTPT

---

**UTPT**

**Upper Student’s t Distribution Command:** Returns the probability \( \text{utpt}(n, x) \) that a Student’s \( t \) random variable is greater than \( x \), where \( n \) is the number of degrees of freedom of the distribution.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( x )</td>
<td>→</td>
<td>utpt(n, x)</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \text{MTH NXT PROB NXT UTPT} \)
UTPT

Affected by Flags: None

Remarks: The following is the defining equation for all $x$.

$$utpt(n, x) = \left[ \frac{\Gamma \left( \frac{n+1}{2} \right)}{\Gamma \left( \frac{n}{2} \right) \sqrt{n\pi}} \right] \int_x^\infty \left( 1 + \frac{t^2}{n} \right)^{-\frac{n+1}{2}} dt$$

For any value $z$, $\Gamma \left( \frac{z}{2} \right) = \left( \frac{z}{2} - 1 \right)!$, where $!$ is the HP 48 factorial command.

The value $n$ is rounded to the nearest integer and, when rounded, must be positive.

Related Commands: UTPC, UTPF, UTPN

UVAL

Unit Value Function: Returns the numerical part of a unit object.

<table>
<thead>
<tr>
<th>UVAL</th>
<th>Unit Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>$x_{\text{unit}}$</td>
<td>$x$</td>
</tr>
<tr>
<td>Level 1</td>
<td>'symb'</td>
<td>'UVAL(symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access: «)(UNITS) U

Affected by Flags: Numerical Results (–3)

Related Commands: CONVERT, UBASE, UFACT, →UNIT
**VAR**

**Variance Command:** Calculates the sample variance of the coordinate values in each of the \( m \) columns in the current statistics matrix (\( \Sigma \text{DAT} \)).

\[
\text{Variance Command:} \quad \text{Calculates the sample variance of the coordinate values in each of the } m \text{ columns in the current statistics matrix (} \Sigma \text{DAT}).
\]

- **Level 1** → **Level 1**
  - \( x_{\text{variance}} \)
  - \( [ x_{\text{variance1}} \ldots x_{\text{variance}m} ] \)

**Keyboard Access:** (STAT) 1

**Affected by Flags:** None

**Remarks:** The variance (equal to the square of the standard deviation) is returned as a vector of \( m \) real numbers, or as a single real number if \( m = 1 \). The variances are computed using this formula:

\[
\frac{1}{n - 1} \sum_{i=1}^{n} (x_i - \bar{x})^2
\]

where \( x_i \) is the \( i \)th coordinate value in a column, \( \bar{x} \) is the mean of the data in this column, and \( n \) is the number of data points.

**Related Commands:** MAX\( \Sigma \), MEAN, MIN\( \Sigma \), PSDEV, PVAR, SDEV, TOT

---

**VARS**

**Variables Command:** Returns a list of all variables’ names in the VAR menu (the current directory).

- **Level 1** → **Level 1**
  - \( \{ \text{global}_1 \ldots \text{global}_n \} \)
**VARS**

Keyboard Access: ➡️(MEMORY) ➡️DIR ➡️VARS

Affected by Flags: None

Related Commands: ORDER, PVARS, TVARS

---

**VERSION**

Software Version Command: Displays the software version and copyright message.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>&quot;version number&quot;</td>
<td>&quot;copyright message&quot;</td>
</tr>
</tbody>
</table>

Keyboard Access: None. Must be typed in.

Affected by Flags: None

---

**VTYPE**

Variable Type Command: Returns the type number of the object contained in the named variable.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td>( n_{type} )</td>
</tr>
<tr>
<td>( n_{port} : name_{backup} )</td>
<td>→</td>
<td>( n_{type} )</td>
</tr>
<tr>
<td>( n_{port} : n_{library} )</td>
<td>→</td>
<td>( n_{type} )</td>
</tr>
</tbody>
</table>

Keyboard Access: PRG ➡️ TYPE ➡️ NXT ➡️ NXT ➡️ VTYPE

Affected by Flags: None
Remarks: If the named variable does not exist, VTYPE returns -1. For a table of the objects' type numbers, see the entry for TYPE.

Related Commands: TYPE

→V2

Stack to Vector/Complex Number Command: Converts two numbers from the stack into a 2-element vector or a complex number.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td></td>
<td>[ x y ]</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td></td>
<td>[ x Δy ]</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td></td>
<td>(x, y)</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td></td>
<td>(x, Δy)</td>
</tr>
</tbody>
</table>

Keyboard Access: [MTH] VECTR →V2

Affected by Flags: Complex Mode (-19), Coordinate System (-16)

Remarks: The result returned depends on the setting of flags -16 and -19, as shown in the following table:

<table>
<thead>
<tr>
<th>Flag -16 clear (Rectangular mode)</th>
<th>Flag -19 clear</th>
<th>Flag -19 set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag -16 set (Polar mode)</td>
<td>[ x y ]</td>
<td>(x, y)</td>
</tr>
<tr>
<td>Flag -16 set (Polar mode)</td>
<td>[ x Δy ]</td>
<td>(x, Δy)</td>
</tr>
</tbody>
</table>

Examples: With flag -19 clear, and flags -16 clear, 2 3 →V2 returns [ 2 3 ].

With flag -19 set and flag -16 set (Polar/Spherical mode), 2 3 →V2 returns (2, Δ3).
$\rightarrow \text{V2}$

**Related Commands:** $V\rightarrow$, $\rightarrow \text{V3}$

$\rightarrow \text{V3}$

**Stack to 3-Element Vector Command:** Converts three numbers into a 3-element vector.

---

### Table: Conversion of Three Numbers into a 3-Element Vector

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$\rightarrow$</td>
<td>$[ x_1 \ x_2 \ x_3 ]$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$x_{\text{theta}}$</td>
<td>$x_2$</td>
<td>$\rightarrow$</td>
<td>$[ x_1 \ \Delta x_{\text{theta}} \ x_2 ]$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$x_{\text{theta}}$</td>
<td>$x_{\text{phi}}$</td>
<td>$\rightarrow$</td>
<td>$[ x_1 \ \Delta x_{\text{theta}} \ \Delta x_{\text{phi}} ]$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** $\text{MTH} \ \text{VECTR} \ \rightarrow \text{V3}$

**Affected by Flags:** Coordinate System ($-15$ and $-16$)

**Remarks:** The result returned depends on the coordinate mode used, as shown in the following table:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular</td>
<td>$[ x_1 \ x_2 \ x_3 ]$</td>
</tr>
<tr>
<td>(flag $-16$ clear)</td>
<td></td>
</tr>
<tr>
<td>Polar/Cylindrical</td>
<td>$[ x_1 \ x \Delta_{\text{theta}} \ x_2 ]$</td>
</tr>
<tr>
<td>(flag $-15$ clear and $-16$ set)</td>
<td></td>
</tr>
<tr>
<td>Polar/Spherical</td>
<td>$[ x_1 \ x \Delta_{\text{theta}} \ x \Delta_{\text{phi}} ]$</td>
</tr>
<tr>
<td>(flag $-15$ and $-16$ set)</td>
<td></td>
</tr>
</tbody>
</table>

**Examples:** With flag $-16$ clear (Rectangular mode), $1 \ 2 \ 3 \ \rightarrow \text{V3}$ returns $[ 1 \ 2 \ 3 ]$.

With flag $-15$ clear and $-16$ set (Polar/Cylindrical mode), $1 \ 2 \ 3 \ \rightarrow \text{V3}$ returns $[ 1 \ 2 \ 3 ]$.  

3-370 Command Reference
With flags $-15$ and $-16$ set (Polar/Spherical mode), $1 \rightarrow 3 \rightarrow V \rightarrow$ returns $[1 \leq 2 \leq 3]$.

**Related Commands:** $V \rightarrow$, $\rightarrow V^2$

---

**$V \rightarrow$**

**Vector/Complex Number to Stack Command:** Separates a vector or complex number into its component elements.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level n .. Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x \ y]$</td>
<td>$\rightarrow$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$[x_r \Delta y_{\theta}]$</td>
<td>$\rightarrow$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$[x_1 \ x_2 \ x_3]$</td>
<td>$\rightarrow$</td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
</tr>
<tr>
<td>$[x_1 \Delta x_{\theta} \ x_2]$</td>
<td>$\rightarrow$</td>
<td>$x_1$</td>
<td>$x_{\theta}$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>$[x_1 \Delta x_{\theta} \Delta x_{\phi}]$</td>
<td>$\rightarrow$</td>
<td>$x_1$</td>
<td>$x_{\theta}$</td>
<td>$x_{\phi}$</td>
</tr>
<tr>
<td>$[x_1 \ x_2 \ldots \ x_n]$</td>
<td>$\rightarrow$</td>
<td>$x_1 \ldots x_{n-2}$</td>
<td>$x_{n-1}$</td>
<td>$x_n$</td>
</tr>
<tr>
<td>$(x, y)$</td>
<td>$\rightarrow$</td>
<td>$x$</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>$(x_r, \Delta y_{\theta})$</td>
<td>$\rightarrow$</td>
<td>$x_r$</td>
<td>$y_{\theta}$</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** $(\text{MTH}) \text{VECTR} \rightarrow V \rightarrow$

**Affected by Flags:** Coordinate System ($-15$ and $-16$)

The elements of the argument complex number or vector are converted from their values in Rectangular mode (the form in which the complex number or vector is stored internally) to the current coordinate system mode before being returned to the stack. This means that the element values returned to the stack always match the displayed element values of the argument vector or complex number.

**Remarks:** For vectors with four or more elements, $V \rightarrow$ executes independently of the coordinate system mode, and always returns the elements of the vector to the stack as they are stored internally (in rectangular form). Thus, $V \rightarrow$ is equivalent to OBJ$\rightarrow$ for vectors with four or more elements.
Examples: With flag -16 clear (Rectangular mode), \( (2, 3) \rightarrow \)
returns 2 to level 2 and 3 to level 1.

With flag -15 clear and flag -16 set (Polar/Cylindrical mode),
\( (2 \theta 4 7) \rightarrow \) returns 2 to level 3, 7 to level 2, and 4 to level 1.

\( (9 7 5 3) \rightarrow \) returns 9 to level 4, 7 to level 3, 5 to level 2, and 3 to level 1, independent of the state of flags -15 and -16.

Related Commands: \( \rightarrow V2, \rightarrow V3 \)

*

Multiply Width Command: Multiplies a plot’s horizontal scale by \( x_{\text{factor}} \).

<table>
<thead>
<tr>
<th>Level 1 ( \rightarrow ) Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{\text{factor}} ) ( \rightarrow )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \langle \text{PLOT} \rangle \text{PPAR} \text{NXT} \rightarrow W \)

Affected by Flags: None

Remarks: Executing \( *W \) changes the \( x \)-axis display range (\( x_{\text{min}} \) and \( x_{\text{max}} \) in the reserved variable \( PPAR \)). The plot center (the user-unit coordinate of the center pixel) does not change.

Related Commands: AUTO, \( \star H \), XRNG
WAIT

Wait Command: Suspends program execution for specified time, or until a key is pressed.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>→</td>
<td>$x_{\text{key}}$</td>
</tr>
<tr>
<td>-1</td>
<td>→</td>
<td>$x_{\text{key}}$</td>
</tr>
</tbody>
</table>

Keyboard Access: PRG NXT IN WAIT

Affected by Flags: None

Remarks: The function of WAIT depends on the argument, as follows:

- Argument $x$ interrupts program execution for $x$ seconds.
- Argument 0 suspends program execution until a valid key is pressed (see below). WAIT then returns $x_{\text{key}}$, which defines where the pressed key is on the keyboard, and resumes program execution.

$x_{\text{key}}$ is a three-digit number that identifies a key's location on the keyboard. See the entry for ASN for a description of the format of $x_{\text{key}}$.

- Argument -1 works as with argument 0, except that the currently specified menu is also displayed.

 joked 23, 23, 0, 3, 23, and 23 4 are not by themselves valid keys.

Arguments 0 or -1 do not affect the display, so that messages persist even though the keyboard is ready for input (FREEZE is not required).

Normally, the MENU command does not update the menu keys until a program halts or ends. WAIT with argument -1 enables a previous execution of MENU to display that menu while the program is suspended for a key press.
WAIT

Examples: This program:

```
« "Press [1] to add Press any other key to subtract" 1 DISP 0 WAIT IF 82.1 SAME THEN + ELSE − END »
```
displays a prompting message and halts program execution until a key is pressed. If the 1 key (location 82.1) is pressed, two numbers on the stack are added. If any other key is pressed, two numbers on the stack are subtracted.

This program:

```
« { ADD ( ) ( ) ( ) SUB } MENU "Press [ADD] to add Press [SUB] to subtract" 1 DISP −1 WAIT IF 11.1 SAME THEN + ELSE − END »
```
builds a custom menu with labels ADD and SUB and a prompting message. Executing −1 WAIT displays the custom menu (note that it’s not active) and suspends execution for keyboard input. If the ADD menu key (location 11.1) is pressed, two numbers on the stack are added. If any other key is pressed, two numbers on the stack are subtracted.

Related Commands: KEY

WHILE

WHILE Indefinite Loop Structure Command: Starts the WHILE ... REPEAT ... END indefinite loop structure.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>−</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHILE</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td>REPEAT</td>
<td>T/F</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>−</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: PRG BRCH WHILE WHILE

Affected by Flags: None

3-374 Command Reference
Remarks: WHILE ... REPEAT ... END repeatedly evaluates a test and executes a loop clause if the test is true. Since the test clause occurs before the loop-clause, the loop clause is never executed if the test is initially false. The syntax is this:

\[
\text{WHILE test-clause REPEAT loop-clause END}
\]

The test clause is executed and must return a test result to the stack. REPEAT takes the value from the stack. If the value is not zero, execution continues with the loop clause; otherwise, execution resumes following END.

Related Commands: DO, END, REPEAT

### WIREFRAME

**WIREFRAME Plot Type Command:** Sets the plot type to WIREFRAME.

**Keyboard Access:** 

[249x551]WIREFRAME

**Affected by Flags:** None

**Remarks:** When the plot type is set to WIREFRAME, the DRAW command plots a perspective view of the graph of a scalar function of two variables. WIREFRAME requires values in the reserved variables EQ, VPAR, and PPAR.

VPAR has the following form:

\[
\{ x_{\text{left}} \ x_{\text{right}} \ y_{\text{near}} \ y_{\text{far}} \ z_{\text{low}} \ z_{\text{high}} \ x_{\text{min}} \ x_{\text{max}} \ y_{\text{min}} \ y_{\text{max}} \ x_{\text{eye}} \\
\ y_{\text{eye}} \ z_{\text{eye}} \ x_{\text{step}} \ y_{\text{step}} \ z_{\text{step}} \}
\]

For plot type WIREFRAME, the elements of VPAR are used as follows:

- \(x_{\text{left}}\) and \(x_{\text{right}}\) are real numbers that specify the width of the view space.
- \(y_{\text{near}}\) and \(y_{\text{far}}\) are real numbers that specify the depth of the view space.
- \(z_{\text{low}}\) and \(z_{\text{high}}\) are real numbers that specify the height of the view space.
WIREFRAME

- \( x_{\text{min}} \) and \( x_{\text{max}} \) are not used.
- \( y_{\text{min}} \) and \( y_{\text{max}} \) are not used.
- \( x_{\text{eye}}, y_{\text{eye}}, \) and \( z_{\text{eye}} \) are real numbers that specify the point in space from which the graph is viewed.
- \( x_{\text{step}} \) and \( y_{\text{step}} \) are real numbers that set the number of \( x \)-coordinates versus the number of \( y \)-coordinates plotted.

The plotting parameters are specified in the reserved variable \( PPAR \), which has this form:

\[
\langle (x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}}), \text{indep}, \text{res}, \text{axes}, \text{ptype}, \text{depend} \rangle
\]

For plot type WIREFRAME, the elements of \( PPAR \) are used as follows:

- \( (x_{\text{min}}, y_{\text{min}}) \) is not used.
- \( (x_{\text{max}}, y_{\text{max}}) \) is not used.
- \( \text{indep} \) is a name specifying the independent variable. The default value of \( \text{indep} \) is \( X \).
- \( \text{res} \) is not used.
- \( \text{axes} \) is not used.
- \( \text{ptype} \) is a name specifying the plot type. Executing the command WIREFRAME places the command name WIREFRAME in \( \text{ptype} \).
- \( \text{depend} \) is a name specifying the dependent variable. The default value is \( Y \).

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, YSLICE
**Warmstart Log Command:** Returns four strings recording the date, time, and cause of the four most recent warmstart events.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 4 ... Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;log_4&quot; ... &quot;log_1&quot;</td>
</tr>
</tbody>
</table>

**Keyboard Access:** None. Must be typed in.

**Affected by Flags:** Date Format (−42)

**Remarks:** Each string "log_n" has the form "code–date time". The following table summarizes the legal values of code and their meanings.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ø</td>
<td>The warmstart log was cleared by pressing [ON] [SPC] and then [ON] to wake the calculator up. [ON] [SPC] puts the HP 48 in “Coma mode” (very low power with the system clock stopped). Pressing [ON] then clears the log and warmstarts the system.</td>
</tr>
<tr>
<td>1</td>
<td>The interrupt system detected a very low battery condition at the battery contacts (not the same as a low system voltage), and put the calculator in a “Deep Sleep mode” (with the system clock running). When [ON] is pressed after the battery voltage is restored, the system warmstarts and puts a 1 in the log.</td>
</tr>
<tr>
<td>2</td>
<td>Hardware failed during IR transmission (timeout).</td>
</tr>
<tr>
<td>3</td>
<td>Run through address 0.</td>
</tr>
<tr>
<td>4</td>
<td>System time is corrupt.</td>
</tr>
<tr>
<td>5</td>
<td>A Deep Sleep wakeup (for example, [ON], Alarm) detected no change to port status, but some changes in data on one or both cards.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>6</td>
<td>Unused.</td>
</tr>
<tr>
<td>7</td>
<td>A 5-nibble word (CMOS test word) in RAM was corrupt. (This word is checked on every interrupt, but it is used only as an indicator of potentially corrupt RAM.)</td>
</tr>
</tbody>
</table>
| 8    | One of the following anomalies involving device configuration was detected:  
  - The interrupt system detected that one of the five devices was not configured.  
  - During a warmstart, an unexpected device ID chain was encountered while attempting to configure 3 (Port1, Port2, Xtra) of the 5 devices.  
  - Same as previous, but detected during Deep Sleep wakeup. |
| 9    | The alarm list is corrupt. |
| A    | Unused.     |
| B    | The card module was removed (or card bounce). |
| C    | Hardware reset occurred (for example, an electrostatic-discharge or user reset). |
| D    | An expected System (RPL) error handler was not found in runstream. |
| E    | The configuration table is corrupt (bad checksum for table data). |
| F    | The system RAM card was removed. |

The date and time stamp (*date time*) part of the log may be displayed as `00...0000` for one of three reasons:  
- The system time was corrupt when the stamp was recorded.  
- The date and time stamp itself is corrupt (bad checksum).  
- Fewer than four warmstarts have occurred since the log was last cleared.
**ΣX**

**Sum of x-Values Command:** Sums the values in the independent-variable column of the current statistical matrix (reserved variable ΣDAT).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>ΣXsum</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \mathbb{C} \text{STAT} \) \( \text{SUMS} \) \( \Sigma X \)

**Affected by Flags:** None

**Remarks:** The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable ΣPAR. The default independent-variable column number is 1.

**Related Commands:** NΣ, XCOL, ΣX*Y, ΣX\(^2\), ΣY, ΣY\(^2\)

---

**ΣX^2**

**Sum of Squares of x-Values Command:** Sums the squares of the values in the independent-variable column of the current statistical matrix (reserved variable ΣDAT).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>ΣXsum</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \mathbb{C} \text{STAT} \) \( \text{SUMS} \) \( ΣX^2 \)

**Affected by Flags:** None

**Remarks:** The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable ΣPAR. The default independent-variable column number is 1.
Related Commands:  $\Sigma \Sigma, \Sigma X, XCOL, \Sigma X+Y, \Sigma Y, \Sigma Y^2$

**XCOL**

**Independent Column Command:** Specifies the independent-variable column of the current statistics matrix (reserved variable $\Sigma DAT$).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>$\rightarrow$</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{col}$</td>
<td>$\rightarrow$</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** 

的相关命令:  BARPLOT, BESTFIT, COL$\Sigma$, CORR, COV, 
EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, 
PWRFIT, SCATR PLOT, YCOL

**Affected by Flags:** None

**Remarks:** The independent-variable column number is stored as the first parameter in the reserved variable $\Sigma PAR$. The default independent-variable column number is 1.

XCOL will accept a noninteger real number and store it in $\Sigma PAR$, but subsequent commands that utilize the XCOL specification in $\Sigma PAR$ will cause an error.
**XMIT**

**Serial Transmit Command:** Sends a string serially without using Kermit protocol, and returns a single digit that indicates whether the transmission was successful.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;string&quot;</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>&quot;string&quot;</td>
<td>&quot;substring_unsent&quot;</td>
<td>0</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \(\uparrow\) I/O \(\uparrow\) NXT \(\uparrow\) SERIA \(\uparrow\) XMIT

**Affected by Flags:** I/O Device (-33)

**Remarks:** XMIT is useful for communicating with non-Kermit devices such as RS-232 printers.

If the transmission is successful, XMIT returns a 1. If the transmission is not successful, XMIT returns the unsent portion of the string and a Ø. Use ERRM to get the error message.

After receiving an XOFF command (with transmit pacing in the reserved variable \(IOPAR\) set), XMIT stops transmitting and waits for an XON command. XMIT resumes transmitting if an XON is received before the time-out set by STIME elapses; otherwise, XMIT terminates, returns a Ø, and stores "Timeout" in ERRM.

**Related Commands:** BUFLEN, SBRK, SRECV, STIME
**XOR**

**Exclusive OR Function:** Returns the logical exclusive OR of two arguments.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( #n_1 )</td>
<td>( #n_2 )</td>
<td>( \rightarrow )</td>
<td>( #n_3 )</td>
</tr>
<tr>
<td>&quot;string(_1)&quot;</td>
<td>&quot;string(_2)&quot;</td>
<td>( \rightarrow )</td>
<td>&quot;string(_3)&quot;</td>
</tr>
<tr>
<td>( T/F_1 )</td>
<td>( T/F_2 )</td>
<td>( \rightarrow )</td>
<td>0/1</td>
</tr>
<tr>
<td>( 'symb' )</td>
<td>( T/F )</td>
<td>( \rightarrow )</td>
<td>( 'symb XOR T/F' )</td>
</tr>
<tr>
<td>( 'symb(_1) )</td>
<td>( 'symb(_2) )</td>
<td>( \rightarrow )</td>
<td>( 'symb(_1) XOR symb(_2)' )</td>
</tr>
</tbody>
</table>

**Keyboard Access:**

MTH BASE NXT LOGIC XOR

PRG TEST NXT XOR

**Affected by Flags:** Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

**Remarks:** When the arguments are binary integers or strings, XOR does a bit-by-bit (base 2) logical comparison:

- Binary integer arguments are treated as sequences of bits with length equal to the current wordsize. Each bit in the result is determined by comparing the corresponding bits (\( bit_1 \) and \( bit_2 \)) in the two arguments, as shown in the following table.

<table>
<thead>
<tr>
<th>( bit_1 )</th>
<th>( bit_2 )</th>
<th>( bit_1 XOR bit_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
String arguments are treated as sequences of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must be the same length.

When the arguments are real numbers or symbolics, XOR simply does a true/false test. The result is 1 (true) if either, but not both, arguments are nonzero; it is 0 (false) if both arguments are nonzero or zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic objects, then the result is an algebraic of the form \(' symb_1 \ XOR \ symb_2 \'. Execute \(\rightarrow \text{NUM} \) (or set flag \(-3\) before executing XOR) to produce a numeric result from the algebraic result.

Related Commands: AND, NOT, OR

---

**XPON**

**Exponent Function:** Returns the exponent of the argument.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>(\rightarrow)</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>(\rightarrow)</td>
<td>(n_{\text{expon}})</td>
</tr>
<tr>
<td>'symb'</td>
<td>(\rightarrow)</td>
<td>'XPON(symb)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH REAL \(\rightarrow\) XPON

**Affected by Flags:** Numerical Results \((-3)\)

**Examples:**
1. \(1.2\times10^3 \text{XPON}\) returns 34.
2. \(1.2\times10^3 \text{XPON}\) returns 4.
3. \('\text{A}\times10^3 \text{XPON}\) returns 'XPON('A\times10^3')'.

**Related Commands:** MANT, SIGN
XRECV

XModem Receive Command: Prepares the HP 48 to receive an object via XModem. The received object is stored in the given variable name.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access:  

Affected by Flags: I/O Device (−33), RECV Overwrite (−36)

Remarks: The transfer will start more quickly if you start the XModem sender before executing XRECV.

Invalid object names cause an error. If flag −36 is clear, object names that are already in use also cause an error.

If you are transferring data between two HP 48s, executing \{\text{AAA BBB CCC}\} XRECV receives AAA, BBB, and CCC. You also need to use a list on the sending end (\{\text{AAA BBB CCC}\} XSEND, for example).

Related Commands: BAUD, RECV, RECN, SEND, XSEND

XRNG

x-Axis Display Range Command: Specifies the x-axis display range.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{\text{min}}$</td>
<td>$x_{\text{max}}$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access:  

3-384 Command Reference
**Affected by Flags:** None

**Remarks:** The $x$-axis display range is stored in the reserved variable `PPAR` as $x_{\text{min}}$ and $x_{\text{max}}$ in the complex numbers $(x_{\text{min}}, y_{\text{min}})$ and $(x_{\text{max}}, y_{\text{max}})$. These complex numbers are the first two elements of `PPAR` and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of $x_{\text{min}}$ and $x_{\text{max}}$ are $-6.5$ and $6.5$, respectively.

**Related Commands:** AUTO, PDIM, PMAX, PMIN, YRNG

---

**XROOT**

**xth Root of y Command:** Computes the $x$th root of a real number.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>$x$</td>
<td>→</td>
<td>$\sqrt[y]{y}$</td>
</tr>
<tr>
<td>$\text{'symb}_1$'</td>
<td>$\text{'symb}_2$'</td>
<td>→</td>
<td>'XROOT($\text{'symb}_2$, $\text{'symb}_1$)'</td>
</tr>
<tr>
<td>$\text{'symb'}$</td>
<td>$x$</td>
<td>→</td>
<td>'XROOT($x$, $\text{'symb'}$)'</td>
</tr>
<tr>
<td>$y$</td>
<td>$\text{'symb'}$</td>
<td>→</td>
<td>'XROOT($\text{'symb'}$, $y$)'</td>
</tr>
<tr>
<td>$y_{\text{unit}}$</td>
<td>$x$</td>
<td>→</td>
<td>$\sqrt[y_{\text{unit}}]{x}$</td>
</tr>
<tr>
<td>$y_{\text{unit}}$</td>
<td>$\text{'symb'}$</td>
<td>→</td>
<td>'XROOT($\text{'symb'}$, $y_{\text{unit}}$)'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** $→\sqrt[y]{y}$

**Affected by Flags:** Numerical Results (-3)

**Remarks:** Note that while the stack syntax is $y$ $x$ XROOT (the root is the second argument), the algebraic syntax is XROOT($x$, $y$) (the root is the first argument) for consistency with the EquationWriter application.

XROOT is equivalent to $y^{1/x}$, but with greater accuracy.

If $y < 0$, $x$ must be an integer.

**Related Commands:**
**XSEND**

**XModem Send Command:** Sends a copy of the named object via XModem.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \leftarrow \) I/O (NXT) \( \rightarrow \) XSEND

**Affected by Flags:** I/O Device (-33)

**Remarks:** A receiving HP 48 must execute XRECV to receive an object via XModem.

To start the transfer more quickly, start the receiving XModem after executing XSEND. Also, configuring the receiving modem not to do CRC checksums (if possible) will avoid a 30 to 60-second delay when starting the transfer.

If you are transferring data between two HP 48s, executing \( \langle \text{AAA BBB CCC} \rangle \) XSEND sends AAA, BBB, and CCC. You also need to use a list on the receiving end (\( \langle \text{AAA BBB CCC} \rangle \) XRECV, for example).

**Related Commands:** BAUD, RECN, RECV, SEND, XRECV

**XVOL**

**X Volume Coordinates Command:** Sets the width of the view volume in the reserved variable VPAR.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{\text{left}} )</td>
<td>( x_{\text{right}} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

3-386 Command Reference
**XXRNG**

**X Range of an Input Plane (Domain) Command:** Specifies the x range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{\text{min}}$</td>
<td>$x_{\text{max}}$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** 🖨 PLOT NXT 3D VPAR XWOL

**Affected by Flags:** None

**Remarks:** $x_{\text{left}}$ and $x_{\text{right}}$ set the x-coordinates for the view volume used in 3D plots. These values are stored in the reserved variable $VPAR$. See appendix D, "Reserved Variables," for more information about $VPAR$.

**Related Commands:** EYEPT, XXRNG, YWOL, YYRNG, ZWOL
**ΣX*Y**

**Sum of x Times y Command:** Sums the products of each of the corresponding values in the independent- and dependent-variable columns of the current statistical matrix (reserved variable ΣDAT).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>x_{\text{sum}}</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \leftarrow \text{(STAT)} \sum \text{ΣX*Y} \)

**Affected by Flags:** None

**Remarks:** The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable ΣPAR. The default independent-variable column number is 1.

The dependent-variable column is specified by YCOL and is stored as the second parameter in reserved variable ΣPAR. The default dependent-variable column number is 2.

**Related Commands:** ΝΣ, ΣΧ, ΧCOL, ΣΧ^2, ΣY, ΣY^2

---

**ΣY**

**Sum of y-Values Command:** Sums the values in the dependent variable column of the current statistical matrix (reserved variable ΣDAT).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>x_{\text{sum}}</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \leftarrow \text{(STAT)} \sum \text{ΣY} \)

**Affected by Flags:** None

3-388 Command Reference
Remarks: The dependent variable column is specified by YCOL, and is stored as the second parameter in the reserved variable \( \Sigma PAR \). The default dependent variable column number is 2.

Related Commands: \( \Sigma X, \Sigma X, XCOL, \Sigma X * Y, \Sigma X^2, YCOL, \Sigma Y^2 \)

\( \Sigma Y^2 \)

Sum of Squares of y-Values Command: Sums the squares of the values in the dependent variable column of the current statistical matrix (reserved variable \( \Sigma DAT \)).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>( x_{sum} )</td>
</tr>
</tbody>
</table>

Keyboard Access: (STAT) \( \Sigma \) SUMS \( \Sigma Y^2 \)

Affected by Flags: None

Remarks: The dependent variable column is specified by YCOL. The default dependent variable column number is 2.

Related Commands: \( \Sigma X, \Sigma X, XCOL, \Sigma X * Y, \Sigma X^2, YCOL, \Sigma Y \)

\( YCOL \)

Dependent Column Command: Specifies the dependent variable column of the current statistics matrix (reserved variable \( \Sigma DAT \)).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{col} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>
**YCOL**

**Keyboard Access:** \( \text{STAT} \) \( \Sigma PAR \) \( YCOL \)

**Affected by Flags:** None

**Remarks:** The dependent variable column number is stored as the second parameter in the reserved variable \( \Sigma PAR \). The default dependent variable column number is 2.

YCOL will accept a noninteger real number and store it in \( \Sigma PAR \), but subsequent commands that utilize the YCOL specification in \( \Sigma PAR \) will cause an error.

**Related Commands:** BARPLOT, BESTFIT, COLΣ, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL

---

**YRNG**

**y-Axis Display Range Command:** Specifies the y-axis display range.

| Level 2 | Level 1 | \( \rightarrow \) | Level 1 |
|---------|---------|------------------|
| \( y_{min} \) | \( y_{max} \) | \( \rightarrow \) |

**Keyboard Access:** \( \text{PLOT} \) \( PPAR \) \( YRNG \)

**Affected by Flags:** None

**Remarks:** The y-axis display range is stored in the reserved variable \( PPAR \) as \( y_{min} \) and \( y_{max} \) in the complex numbers \( \langle x_{min}, y_{min} \rangle \) and \( \langle x_{max}, y_{max} \rangle \). These complex numbers are the first two elements of \( PPAR \) and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of \( y_{min} \) and \( y_{max} \) are -3.1 and 3.2, respectively.

**Related Commands:** AUTO, PDIM, PMAX, PMIN, XRNG
**YSLICE**

**Y-Slice Plot Command:** Sets the plot type to YSLICE.

**Keyboard Access:** \( \text{PLOT} \text{ NXT} \text{ 3D PTYPE YSLICE} \)

**Affected by Flags:** None

**Remarks:** When plot type is set YSLICE, the DRAW command plots a slicing view of a scalar function of two variables. YSLICE requires values in the reserved variables \( EQ \), \( VPAR \), and \( PPAR \).

\( VPAR \) has the following form:

\[
\{ x_{\text{left}}, x_{\text{right}}, y_{\text{near}}, y_{\text{far}}, z_{\text{low}}, z_{\text{high}}, x_{\min}, x_{\max}, y_{\min}, y_{\max}, x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}}, x_{\text{step}}, y_{\text{step}} \}
\]

For plot type YSLICE, the elements of \( VPAR \) are used as follows:

- \( x_{\text{left}} \) and \( x_{\text{right}} \) are real numbers that specify the width of the view space.
- \( y_{\text{near}} \) and \( y_{\text{far}} \) are real numbers that specify the depth of the view space.
- \( z_{\text{low}} \) and \( z_{\text{high}} \) are real numbers that specify the height of the view space.
- \( x_{\min} \) and \( x_{\max} \) are not used.
- \( y_{\min} \) and \( y_{\max} \) are not used.
- \( x_{\text{eye}}, y_{\text{eye}}, \) and \( z_{\text{eye}} \) are real numbers that specify the point in space from which the graph is viewed.
- \( x_{\text{step}} \) determines the interval between plotted x-values within each "slice".
- \( y_{\text{step}} \) determines the number of slices to draw.

The plotting parameters are specified in the reserved variable \( PPAR \), which has this form:

\[
\{ (x_{\min}, y_{\min}), (x_{\max}, y_{\max}) \text{ indep res axes ptype depend} \}
\]

For plot type YSLICE, the elements of \( PPAR \) are used as follows:

- \( (x_{\min}, y_{\min}) \) is not used.
- \( (x_{\max}, y_{\max}) \) is not used.
**YSLICE**

- `indep` is a name specifying the independent variable. The default value of `indep` is `X`.

- `res` is a real number specifying the interval, in user-unit coordinates, between plotted values of the independent variable; or a binary integer specifying the interval in pixels. The default value is `0`, which specifies an interval of 1 pixel.

- `axes` is not used.

- `ptype` is a command name specifying the plot type. Executing the command YSLICE places YSLICE in `ptype`.

- `depend` is a name specifying the dependent variable. The default value is `Y`.

**Related Commands:** BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME

---

**YVOL**

**Y Volume Coordinates Command:** Sets the depth of the view volume in the reserved variable `VPAR`.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_{near} )</td>
<td>( y_{far} )</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** ◄ PLOT NXT 3D VPAR YVOL

**Affected by Flags:** None

**Remarks:** The variables \( y_{near} \) and \( y_{far} \) are real numbers that set the y-coordinates for the view volume used in 3D plots. \( y_{near} \) must be less than \( y_{far} \). These values are stored in the reserved variable `VPAR`.

**Related Commands:** EYEPT, XVOL, XXRNG, YYRNG, ZVOL

3-392 Command Reference
ZFACTOR

Gas Compressibility Z Factor Function: Calculates the gas compressibility correction factor for nonideal behavior of a hydrocarbon gas.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_T$</td>
<td>$y_P$</td>
<td>→</td>
<td>$X_{Z\text{factor}}$</td>
</tr>
<tr>
<td>$x_T$</td>
<td>'symb'</td>
<td>→</td>
<td>'ZFACTOR(x_T,symb)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$y_P$</td>
<td>→</td>
<td>'ZFACTOR(symb,y_P)'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>'symb_2'</td>
<td>→</td>
<td>'ZFACTOR(symb_1,symb_2)'</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{EQLIB UTILS ZFACT} \)

Y Range of an Input Plane (Domain) Command: Specifies the y range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{\text{near}}$</td>
<td>$y_{\text{far}}$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{PLOT NXT 3D VPAR YYRN} \)

Affected by Flags: None

Remarks: The variables $y_{\text{near}}$ and $y_{\text{far}}$ are real numbers that set the y-coordinates for the input plane. These values are stored in the reserved variable VPAR.

Related Commands: EYEPT, XVOL, XXRNG, YVOL, ZVOL
ZFACTOR

Affected by Flags: Numerical Results (−3)

Remarks: $x_T$ is the reduced temperature: the ratio of the actual temperature ($T$) to the pseudocritical temperature ($T_c$). (Calculate the ratio using absolute temperatures.) $x_T$ must be between 1.05 and 3.0.

$y_P$ is the reduced pressure: the ratio of the actual pressure ($P$) to the pseudocritical pressure ($P_c$). $y_P$ must be between 0 and 30.

$x_T$ and $y_P$ must be real numbers or unit objects that reduce to dimensionless numbers.

ZVOL

Z Volume Coordinates Command: Sets the height of the view volume in the reserved variable $VPAR$.

Keyboard Access: («)(PLOT)(NXT) =

Affected by Flags: None

Remarks: $x_{\text{low}}$ and $x_{\text{high}}$ are real numbers that set the z-coordinates for the view volume used in 3D plots. These values are stored in the reserved variable $VPAR$.

Related Commands: EYEPT, XVOL, XXRNG, YVOL, YYRNG

3-394 Command Reference
Add Analytic Function: Returns the sum of the arguments.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>$z_2$</td>
<td>→</td>
<td>$z_1 + z_2$</td>
</tr>
<tr>
<td>'symb'</td>
<td>'symb'</td>
<td>→</td>
<td>'symb + z'</td>
</tr>
<tr>
<td>'symb'</td>
<td>'symb'</td>
<td>→</td>
<td>'symb_1 + symb_2'</td>
</tr>
<tr>
<td><code>{list_1}</code></td>
<td><code>{list_2}</code></td>
<td>→</td>
<td><code>{list_1 list_2}</code></td>
</tr>
<tr>
<td>obj_A</td>
<td>obj_A</td>
<td>→</td>
<td><code>{obj_A obj_1 ... obj_n}</code></td>
</tr>
<tr>
<td>obj_1 ... obj_n</td>
<td>obj_A</td>
<td>→</td>
<td><code>{obj_1 ... obj_n obj_A}</code></td>
</tr>
<tr>
<td>&quot;string_1&quot;</td>
<td>&quot;string_2&quot;</td>
<td>→</td>
<td>&quot;string_1 string_2&quot;</td>
</tr>
<tr>
<td>obj</td>
<td>obj</td>
<td>→</td>
<td>&quot;string obj&quot;</td>
</tr>
<tr>
<td>#n_1</td>
<td>n_2</td>
<td>→</td>
<td>#n_3</td>
</tr>
<tr>
<td>n_1</td>
<td>#n_2</td>
<td>→</td>
<td>#n_3</td>
</tr>
<tr>
<td>#n_1</td>
<td>#n_2</td>
<td>→</td>
<td>#n_3</td>
</tr>
<tr>
<td>x_1–unit_1</td>
<td>y_unit_2</td>
<td>→</td>
<td>(x_2+y)–unit_2</td>
</tr>
<tr>
<td>'symb'</td>
<td>x_unit</td>
<td>→</td>
<td>'symb + x_unit'</td>
</tr>
<tr>
<td>x_unit</td>
<td>'symb'</td>
<td>→</td>
<td>'x_unit + symb'</td>
</tr>
<tr>
<td>grob_1</td>
<td>grob_2</td>
<td>→</td>
<td>grob_3</td>
</tr>
</tbody>
</table>

Keyboard Access: 📈

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: The sum of a real number $a$ and a complex number $(x, y)$ is the complex number $(x+a, y)$.

The sum of two complex numbers $(x_1, y_1)$ and $(x_2, y_2)$ is the complex number $(x_1+x_2, y_1+y_2)$.
The sum of a real array and a complex array is a complex array, where each element $x$ of the real array is treated as a complex element $(x, 0)$. The arrays must have the same dimensions.

The sum of a binary integer and a real number is a binary integer that is the sum of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the addition.)

The sum of two binary integers is truncated to the current binary integer wordsize.

The sum of two unit objects is a unit object with the same dimensions as the level 1 argument. The units of the two arguments must be consistent.

The sum of two graphics objects is the same as the result of performing a logical OR, except that the two graphics objects must have the same dimensions.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: $0 \, ^\circ\text{C} = 273.15 \, \text{K}$, and $0 \, ^\circ\text{F} = 459.67 \, ^\circ\text{R}$. But when °C or °F represents a difference in thermometer readings, then the temperature is a unit with no additive constant: $1 \, ^\circ\text{C} = 1 \, \text{K}$ and $1 \, ^\circ\text{F} = 1 \, ^\circ\text{R}$.

The calculator assumes that the simple temperature units $x\_\, ^\circ\text{C}$ and $x\_\, ^\circ\text{F}$ represent thermometer temperatures when used as arguments to the functions $\lt$, $\gt$, $\leq$, $\geq$, $\equiv$, and $\neq$. This means that, in order to do the calculation, the calculator will first convert any Celsius temperature to kelvins and any Fahrenheit temperature to Rankines. (For other functions or compound temperature units, such as $x\_\, ^\circ\text{C} / \text{min}$, the calculator assumes temperature units represent temperature differences, so there is no additive constant involved, and hence no conversion.)

The arithmetic operators $+$, $-$, $\%\text{CH}$, and $\%\text{T}$ treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

**Examples:**

```
(1 2 3) (A B C) + returns (1 2 3 A B C).
5_ft 9_in + returns 69_in.
[[0 1][1 3]] [[2 1][0 1]] + returns [[2 2][1 4]].
```
Related Commands: −, *, /, =

Subtract Analytic Function: Returns the difference of the arguments: the object in level 1 is subtracted from the object in level 2.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>$z_2$</td>
<td>−</td>
<td>$z_1 - z_2$</td>
</tr>
<tr>
<td>[array]</td>
<td>[array]</td>
<td>−</td>
<td>[array]_{1-2}</td>
</tr>
<tr>
<td>$z$</td>
<td>'symb'</td>
<td>→</td>
<td>'z - symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$z$</td>
<td>→</td>
<td>'symb - z'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>'symb_2'</td>
<td>→</td>
<td>'symb_1 - symb_2'</td>
</tr>
<tr>
<td>#n_1</td>
<td>n_2</td>
<td>→</td>
<td>#n_3</td>
</tr>
<tr>
<td>n_1</td>
<td>#n_2</td>
<td>→</td>
<td>#n_3</td>
</tr>
<tr>
<td>#n_1</td>
<td>#n_2</td>
<td>→</td>
<td>#n_3</td>
</tr>
<tr>
<td>x_1-unit</td>
<td>y_unit_2</td>
<td>→</td>
<td>(x_2 - y)_unit_2</td>
</tr>
<tr>
<td>'symb'</td>
<td>x_unit</td>
<td>→</td>
<td>'symb - x_unit'</td>
</tr>
<tr>
<td>x_unit</td>
<td>'symb'</td>
<td>→</td>
<td>'x_unit - symb'</td>
</tr>
</tbody>
</table>

Keyboard Access:  □

Affected by Flags: Numerical Results (-3)

Remarks: The difference of a real number $a$ and a complex number $(x, y)$ is $(x - a, y)$ or $(a - x, -y)$. The difference of two complex numbers $(x_1, y_1)$ and $(x_2, y_2)$ is $(x_1 - x_2, y_1 - y_2)$.

The difference of a real array and a complex array is a complex array, where each element $x$ of the real array is treated as a complex element $(x, 0)$. The two array arguments must have the same dimensions.

The difference of a binary integer and a real number is a binary integer that is the sum of the level 2 number and the two’s
complement of the level 1 number. (The real number is converted to a binary integer before the subtraction.)

The difference of two binary integers is a binary integer that is the sum of the level 2 number and the two's complement of the level 1 number.

The difference of two unit objects is a unit object with the same dimensions as the level 1 object. The units of the two arguments must be consistent.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: \( 0 \, ^\circ\text{C} = 273.15 \, \text{K} \), and \( 0 \, ^\circ\text{F} = 459.67 \, ^\circ\text{R} \). But when °C or °F represents a difference in thermometer readings, then the temperature is a unit with no additive constant: \( 1 \, ^\circ\text{C} = 1 \, \text{K} \) and \( 1 \, ^\circ\text{F} = 1 \, ^\circ\text{R} \).

The calculator assumes that the simple temperature units \( x_\circ\text{C} \) and \( x_\circ\text{F} \) represent thermometer temperatures when used as arguments to the functions \(<\), \(>\), \(\leq\), \(\geq\), \(=\), and \(\neq\). This means that, in order to do the calculation, the calculator will first convert any Celsius temperature to kelvins and any Fahrenheit temperature to Rankines. (For other functions or compound temperature units, such as \( x_\circ\text{C}/\text{min} \), the calculator assumes temperature units represent temperature differences, so there is no additive constant involved, and hence no conversion.)

The arithmetic operators +, —, %CH, and %T treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Example: \( 25_\text{ft} 8_\text{in} - \text{returns} \ 292_\text{in} \).

\([\ [[ 5 \ 1 ] [ 3 \ 3 ] \ [ 2 \ 1 ] [ 0 \ 1 ] ] - \text{returns} \ [ [ 3 \ 0 ] [ 3 \ 2 ] ] \).

'TOTAL' 'PART' - returns 'TOTAL-PART'.

Related Commands: \(+, *, /, =\)
Multiply Analytic Function: Returns the product of the arguments.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>z₁</td>
<td>z₂</td>
<td>→</td>
<td>z₁z₂</td>
</tr>
<tr>
<td>[ matrix ]</td>
<td>[ array ]</td>
<td>→</td>
<td>[ matrix x array ]</td>
</tr>
<tr>
<td>z</td>
<td>[ array ]</td>
<td>→</td>
<td>[ array x z ]</td>
</tr>
<tr>
<td>[ array ]</td>
<td>z</td>
<td>→</td>
<td>[ array x z ]</td>
</tr>
<tr>
<td>z</td>
<td>'symb'</td>
<td>→</td>
<td>'z * symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>z</td>
<td>→</td>
<td>'symb * z'</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>'symb₂'</td>
<td>→</td>
<td>'symb₁ * symb₂'</td>
</tr>
<tr>
<td>#n₁</td>
<td>n₂</td>
<td>→</td>
<td>#n₃</td>
</tr>
<tr>
<td>n₁</td>
<td>#n₂</td>
<td>→</td>
<td>#n₃</td>
</tr>
<tr>
<td>#n₁</td>
<td>#n₂</td>
<td>→</td>
<td>#n₃</td>
</tr>
<tr>
<td>x_unit</td>
<td>y_unit</td>
<td>→</td>
<td>x_y_unit</td>
</tr>
<tr>
<td>x</td>
<td>y_unit</td>
<td>→</td>
<td>x_y_unit</td>
</tr>
<tr>
<td>x_unit</td>
<td>y</td>
<td>→</td>
<td>x_y_unit</td>
</tr>
<tr>
<td>'symb'</td>
<td>x_unit</td>
<td>→</td>
<td>'symb * x_unit'</td>
</tr>
<tr>
<td>x_unit</td>
<td>'symb'</td>
<td>→</td>
<td>'x_unit * symb'</td>
</tr>
</tbody>
</table>

Keyboard Access: 

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: The product of a real number $a$ and a complex number $(x, y)$ is the complex number $(xa, ya)$.

The product of two complex numbers $(x₁, y₁)$ and $(x₂, y₂)$ is the complex number $(x₁x₂ − y₁y₂, x₁y₂ + x₂y₁)$.

The product of a real array and a complex array or number is a complex array. Each element $x$ of the real array is treated as a complex element $(x, 0)$.
Multiplying a matrix (level 2) by an array (level 1) returns a matrix product. The matrix must have the same number of columns as the array in level 1 has rows (or elements, if it is a vector).

Although a vector is entered and displayed as a row of numbers, the HP 48 treats a vector as an $n \times 1$ matrix when multiplying matrices or computing matrix norms.

Multiplying a binary integer by a real number returns a binary integer that is the product of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the multiplication.)

The product of two binary integers is truncated to the current binary integer wordsize.

When multiplying two unit objects, the scalar parts and the unit parts are multiplied separately.

**Related Commands:** $+, -, \div, =$
Divide Analytic Function: Returns the quotient of the arguments: the level 2 object divided by the level 1 object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>$z_2$</td>
<td>→</td>
<td>$z_1 / z_2$</td>
</tr>
<tr>
<td>[ array ]</td>
<td>[[ matrix ]]</td>
<td>→</td>
<td>[[ matrix$^{-1}$ x array ]]</td>
</tr>
<tr>
<td>[ array ]</td>
<td>$z$</td>
<td>→</td>
<td>[ array$/$z ]</td>
</tr>
<tr>
<td>$z$</td>
<td>'symb'</td>
<td>→</td>
<td>'z/symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$z$</td>
<td>→</td>
<td>'symb/$z$'</td>
</tr>
<tr>
<td>'symb'</td>
<td>'symb'</td>
<td>→</td>
<td>'symb/symb'</td>
</tr>
<tr>
<td>#n$_1$</td>
<td>n$_2$</td>
<td>→</td>
<td>#n$_3$</td>
</tr>
<tr>
<td>n$_1$</td>
<td>#n$_2$</td>
<td>→</td>
<td>#n$_3$</td>
</tr>
<tr>
<td>#n$_1$</td>
<td>#n$_2$</td>
<td>→</td>
<td>#n$_3$</td>
</tr>
<tr>
<td>x$_1$/unit</td>
<td>y$_1$/unit</td>
<td>→</td>
<td>(x/y)$_1$/unit</td>
</tr>
<tr>
<td>x</td>
<td>y$_1$/unit</td>
<td>→</td>
<td>(x/y)$_1$/unit</td>
</tr>
<tr>
<td>x$_1$/unit</td>
<td>y</td>
<td>→</td>
<td>'symb/x$_1$/unit'</td>
</tr>
<tr>
<td>x$_1$/unit</td>
<td>'symb'</td>
<td>→</td>
<td>'x$_1$/unit/symb'</td>
</tr>
</tbody>
</table>

Keyboard Access:

受影响的标志: 数值结果 (-3)

备注: 一个实数 $a$ 除以一个复数 $(x, y)$ 返回 $(\frac{ax}{x^2+y^2}, \frac{-ay}{x^2+y^2})$。一个复数 $(x, y)$ 除以一个实数 $a$ 返回这个复数 $(x/a , y/a)$。

一个复数 $(x_1, y_1)$ 除以另一个复数 $(x_2, y_2)$ 返回这个复数的商:

Command Reference 3-401
An array $\mathbf{B}$ divided by a matrix $\mathbf{A}$ solves the system of equations $\mathbf{AX}=\mathbf{B}$ for $\mathbf{X}$; that is, $\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$. This operation uses 15-digit internal precision, providing a more precise result than the calculation $\text{INV}(\mathbf{A})\times\mathbf{B}$. The matrix must be square, and must have the same number of columns as the array has rows (or elements, if the array is a vector).

A binary integer divided by a real or binary number returns a binary integer that is the integral part of the quotient. (The real number is converted to a binary integer before the division.) A divisor of zero returns $\# \emptyset$.

When dividing two unit objects, the scalar parts and the unit parts are divided separately.

**Related Commands:** $+,-,*,=$

---

**Power Analytic Function:** Returns the value of the level 2 object raised to the power of the level 1 object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$</td>
<td>$z$</td>
<td>$\rightarrow$</td>
<td>$w^z$</td>
</tr>
<tr>
<td>$z$</td>
<td>'symb'</td>
<td>$\rightarrow$</td>
<td>'z^\text{(symb)}'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$z$</td>
<td>$\rightarrow$</td>
<td>'(symb)^z'</td>
</tr>
<tr>
<td>'symb'</td>
<td>'symb'</td>
<td>$\rightarrow$</td>
<td>'symb_1^\text{(symb}_2)'</td>
</tr>
<tr>
<td>$x$ _unit</td>
<td>$y$</td>
<td>$\rightarrow$</td>
<td>$x^y$ _unit$^y$</td>
</tr>
<tr>
<td>$x$ _unit</td>
<td>'symb'</td>
<td>$\rightarrow$</td>
<td>'(x _unit)^\text{(symb)}$</td>
</tr>
</tbody>
</table>

**Keyboard Access:** $y^z$
**Affected by Flags:** Principal Solution (−1), Numerical Results (−3)

**Remarks:** If either argument is complex, the result is complex.

The branch cuts and inverse relations for $w^z$ are determined by this relationship:

$$w^z = \exp(z \ln w)$$

**Related Commands:** EXP, ISOL, LN, XROOT

---

**Less Than Function:** Tests whether one object is less than another object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$y$</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>#n₁</td>
<td>#n₂</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>&quot;string₁&quot;</td>
<td>&quot;string₂&quot;</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>$x$</td>
<td>'symb'</td>
<td>→</td>
<td>'x&lt;symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$x$</td>
<td>→</td>
<td>'symb&lt;x'</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>'symb₂'</td>
<td>→</td>
<td>'symb₁&lt;symb₂'</td>
</tr>
<tr>
<td>$x_{\text{unit}}₁$</td>
<td>$y_{\text{unit}}₂$</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>$x_{\text{unit}}$</td>
<td>'symb'</td>
<td>→</td>
<td>'x_{\text{unit}}&lt;symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$x_{\text{unit}}$</td>
<td>→</td>
<td>'symb&lt;x_{\text{unit}}'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] TEST

**Affected by Flags:** Numerical Results (−3)

**Remarks:** The function $<$ returns a true test result (1) if the level 2 argument is less than the level 1 argument, or a false test result (0) otherwise.
If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, < returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, “less than” means numerically smaller (1 is less than 2). For real numbers, “less than” also means more negative (−2 is less than −1).

For strings, “less than” means alphabetically previous (“ABC” is less than “DEF”; “AAA” is less than “AAB”; “A” is less than “AA”). In general, characters are ordered according to their character codes. This means, for example, that “B” is less than “a”, since “B” is character code 66, and “a” is character code 97.

For unit objects, the two objects must be dimensionally consistent, and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Related Commands: ≤, >, ≥, ==, ≠

Less Than or Equal Function: Tests whether one object is less than or equal to another object.
I

{ }

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y )</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>( #n_1 )</td>
<td>( #n_2 )</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>( &quot;string_1&quot; )</td>
<td>( &quot;string_2&quot; )</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>( \text{'symb'} )</td>
<td>( x )</td>
<td>→</td>
<td>( \text{'symb} \leq \text{'symb'} )</td>
</tr>
<tr>
<td>( \text{'symb}_1 )</td>
<td>( \text{'symb}_2 )</td>
<td>→</td>
<td>( \text{'symb}_1 \leq \text{'symb}_2 )</td>
</tr>
<tr>
<td>( x_unit_1 )</td>
<td>( y_unit_2 )</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>( x_unit )</td>
<td>( \text{'symb'} )</td>
<td>→</td>
<td>( \text{'x_unit} \leq \text{'symb'} )</td>
</tr>
<tr>
<td>( \text{'symb'} )</td>
<td>( x_unit )</td>
<td>→</td>
<td>( \text{'symb} \leq \text{'x_unit'} )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{PRG TEST} \leq \)

Affected by Flags: Numerical Results (-3)

Remarks: The function \( \leq \) returns a true test result (1) if the level 2 argument is less than or equal to the level 1 argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, \( \leq \) returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, “less than or equal” means numerically equal or smaller (1 is less than 2). For real numbers, “less than or equal” also means equally or more negative (—2 is less than —1).

For strings, “less than or equal” means alphabetically equal or previous (“ABC” is less than or equal to “DEF”; “AAA” is less than or equal to “AAB”; “A” is less than or equal to “AA”). In general, characters are ordered according to their character codes. This means, for example, that “B” is less than “a”, since “B” is character code 66, and “a” is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperature and not differences in temperature. For compound temperature units, the calculator assumes temperature units represent
temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

**Related Commands:** <, >, ≥, ==, ≠

---

**Greater Than Function:** Tests whether one object is greater than another object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>➔</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$y$</td>
<td>➔</td>
<td>0/1</td>
</tr>
<tr>
<td>$#n_1$</td>
<td>$#n_2$</td>
<td>➔</td>
<td>0/1</td>
</tr>
<tr>
<td>&quot;string_1&quot;</td>
<td>&quot;string_2&quot;</td>
<td>➔</td>
<td>0/1</td>
</tr>
<tr>
<td>$x$</td>
<td>'symb'</td>
<td>➔</td>
<td>'x&gt;symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$x$</td>
<td>➔</td>
<td>'symb&gt;x'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>'symb_2'</td>
<td>➔</td>
<td>'symb_1&gt;symb_2'</td>
</tr>
<tr>
<td>$x_unit_1$</td>
<td>$y_unit_2$</td>
<td>➔</td>
<td>0/1</td>
</tr>
<tr>
<td>$x_unit$</td>
<td>'symb'</td>
<td>➔</td>
<td>'x_unit&gt;symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>$x_unit$</td>
<td>➔</td>
<td>'symb&gt;x_unit'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG TEST ➔

**Affected by Flags:** Numerical Results (−3)

**Remarks:** The function > returns a true test result (1) if the level 2 argument is greater than the level 1 argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, > returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "greater than" means numerically greater (2 is greater than 1). For real numbers, "greater than" also means less negative (−1 is greater than −2).
For strings, "greater than" means alphabetically subsequent ("DEF" is greater than "ABC"; "AAB" is greater than "AAA"; "AA" is greater than "A"). In general, characters are ordered according to their character codes. This means, for example, that "a" is greater than "B", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperature. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

**Related Commands:** <, ≤, ≥, ==, ≠

---

**Greater Than or Equal Function:** Tests whether one object is greater than or equal to another object.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>#n₁</td>
<td>#n₂</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>&quot;string₁&quot;</td>
<td>&quot;string₂&quot;</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>x</td>
<td>'symb'</td>
<td>→</td>
<td>'x ≥ symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>x</td>
<td>→</td>
<td>'symb ≥ x'</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>'symb₂'</td>
<td>→</td>
<td>'symb₁ ≥ symb₂'</td>
</tr>
<tr>
<td>x_unit₁</td>
<td>y_unit₂</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>x_unit</td>
<td>'symb'</td>
<td>→</td>
<td>'x_unit ≥ symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>x_unit</td>
<td>→</td>
<td>'symb ≥ x_unit'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** [PRG] TEST ➕

**Affected by Flags:** Numerical Results (-3)
Remarks: The function $\geq$ returns a true test result (1) if the level 2 argument is greater than or equal to the level 1 argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, $\geq$ returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, “greater than or equal to” means numerically equal or greater (2 is greater than or equal to 1). For real numbers, “greater than or equal to” also means equally or less negative (−1 is greater than or equal to −2).

For strings, “greater than or equal to” means alphabetically equal or subsequent (“DEF” is greater than or equal to “ABC”; “AAB” is greater than or equal to “AAA”; “AA” is greater than or equal to “A”). In general, characters are ordered according to their character codes. This means, for example, that “a” is greater than or equal to “B”, since “B” is character code 66, and “a” is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperature. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Related Commands: $<$, $\leq$, $>$, $==$,
The equals sign equates two expressions such that the difference between the two expressions is zero.

In Symbolic Results mode, the result is an algebraic equation. In Numerical Results mode, the result is the difference of the two arguments because = acts equivalent to -. This allows expressions and equations to be used interchangeably as arguments for symbolic and numerical rootfinders.

The numerical evaluation of an equation using the HP Solve application implicitly involves the subtraction of terms. See the entry for "-" for information about the effects of subtraction.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a difference in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The arithmetic operators +, -, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, -, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.
Logical Equality Function: Tests if two objects are equal.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{obj}_1</td>
<td>\textit{obj}_2</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>(x,0)</td>
<td>\textit{x}</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>\textit{x}</td>
<td>(x,0)</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>\textit{z}</td>
<td>'symb'</td>
<td>→</td>
<td>'z==symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>\textit{z}</td>
<td>→</td>
<td>'symb==z'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>'symb_2'</td>
<td>→</td>
<td>'symb_1==symb_2'</td>
</tr>
</tbody>
</table>

Keyboard Access: \texttt{\text{PRG} \text{TEST} \text{==}}

Affected by Flags: Numerical Results (−3)

Remarks: The function \texttt{==} returns a true result (1) if the two objects are the same type and have the same value, or a false result (0) otherwise. Lists and programs are considered to have the same values if the objects they contain are identical.

If one object is algebraic (or a name), and the other is a number (real or complex) or an algebraic, \texttt{==} returns a symbolic comparison expression that can be evaluated to return a test result.

Note that \texttt{==} is used for comparisons, while \texttt{=} separates two sides of an equation.

If the imaginary part of a complex number is 0, it is ignored when the complex number is compared to a real number, so, for example, \texttt{6 (6,0) ==} returns 1.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent 3-410 Command Reference
temperatures and not differences in temperature. For compound
temperature units, the calculator assumes temperature units represent
temperature differences. For more information on using temperature
units with arithmetic functions, refer to the entry for +.

**Related Commands:** SAME, TYPE, <, ≤, >, ≥, ≠

---

**Not Equal Function:** Tests if two objects are not equal.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj₁</td>
<td>obj₂</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>(x,0)</td>
<td>x</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>x</td>
<td>(x,0)</td>
<td>→</td>
<td>0/1</td>
</tr>
<tr>
<td>z</td>
<td>'symb'</td>
<td>→</td>
<td>'z ≠ symb'</td>
</tr>
<tr>
<td>'symb'</td>
<td>z</td>
<td>→</td>
<td>'symb ≠ z'</td>
</tr>
<tr>
<td>'symb₁'</td>
<td>'symb₂'</td>
<td>→</td>
<td>'symb₁ ≠ symb₂'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** PRG TEST ≠

**Affected by Flags:** Numerical Results (-3)

**Remarks:** The function ≠ returns a true result (1) if the two objects
have different values, or a false result (0) otherwise. (Lists and
programs are considered to have the same values if the objects they
contain are identical.)

If one object is algebraic or a name, and the other is a number, a
name, or algebraic, ≠ returns a symbolic comparison expression that
can be evaluated to return a test result.

If the imaginary part of a complex number is 0, it is ignored when
the complex number is compared to real number, so, for example, ∈
(∈, ⊥) ≠ returns ⊥.
For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

**Related Commands:** SAME, TYPE, <, ≤, >, ≥, ==

---

Factorial (Gamma) Function: Returns the factorial \( n! \) of a positive integer argument \( n \), or the gamma function \( \Gamma(x+1) \) of a non-integer argument \( x \).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>→</td>
<td>( n! )</td>
</tr>
<tr>
<td>( x )</td>
<td>→</td>
<td>( \Gamma(x+1) )</td>
</tr>
<tr>
<td>'symb'</td>
<td>→</td>
<td>'(symb)!'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** MTH NXT PROB

**Affected by Flags:** Numerical Results (—3), Underflow Exception (—20), Overflow Exception (—21)

**Remarks:** For \( x \geq 253.1190554375 \) or \( n < 0 \), ! causes an Overflow exception (if flag —21 is set, the exception is treated as an error). For non-integer \( x \leq -254.1082426465 \), ! causes an Underflow exception (if flag —20 is set, the exception is treated as an error).

In algebraic syntax, ! follows its argument. Thus the algebraic syntax for the factorial of 7 is '7!'.

For non-integer arguments \( x \), \( x! = \Gamma(x + 1) \), defined for \( x > -1 \) as this:
\[ \Gamma(x + 1) = \int_0^\infty e^{-t^x} dt \]

and defined for other values of \( x \) by analytic continuation:

\[ \Gamma(x + 1) = n \cdot \Gamma(x) \]

**Related Commands:** COMB, PERM

---

**Integral Function:** Integrates an *integrand* from *lower limit* to *upper limit* with respect to a specified variable of integration.

<table>
<thead>
<tr>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>( \rightarrow )</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower limit</td>
<td>upper limit</td>
<td>integrand</td>
<td>'name'</td>
<td>( \rightarrow )</td>
<td>'symb_integral'</td>
</tr>
</tbody>
</table>

**Keyboard Access:** (\( \rightarrow \))

**Affected by Flags:** Numerical Results (\(-3\)), Number Format (\(-45\) to \(-50\))

**Remarks:** The algebraic syntax for \( \int \) parallels its stack syntax:

\[ \int \text{lower limit, upper limit, integrand, name} \]

where *lower limit*, *upper limit*, and *integrand* can be real or complex numbers, unit objects, names, or algebraic expressions.

Evaluating \( \int \) in Symbolic Results mode (flag \(-3\) clear) returns a symbolic result to level 1. The HP 48 does symbolic integration by *pattern matching*. The HP 48 can integrate the following:

- All built-in functions whose antiderivatives can be expressed in terms of other built-in functions—for example, \( \text{SIN} \) can be integrated since its antiderivative \( \text{COS} \) is a built-in function. The arguments for these functions must be linear.
Sums, differences, and negations of built-in functions whose antiderivatives can be expressed in terms of other built-in functions—for example, \( \sin(x) - \cos(x) \).

Derivatives of all built-in functions—for example, \( \ln(1 + x^2) \) can be integrated because it is the derivative of the built-in function \( \tan \).

Polynomials whose base term is linear—for example, 
\( x^3 + x^2 - 2x + 6 \) can be integrated since \( x \) is a linear term.
\( (x^2 - 6)^3 + (x^2 - 6)^2 \) cannot be integrated since \( x^2 - 6 \) is not linear.

Selected patterns composed of functions whose antiderivatives can be expressed in terms of other built-in functions—for example, 
\( 1/(\cos(x) * \sin(x)) \) returns \( \ln(\tan(x)) \).

If the result of the integration is an expression with no integral sign in the result, the symbolic integration was successful. If, however, the result still contains an integral sign, try rearranging the expression and evaluating again, or estimate the answer using numerical integration.

A successful result of symbolic integration has this form:
\[
\text{result} <\text{name=upper limit}> - <\text{result} <\text{name=lower limit}> >
\]

See the \( | \) (where) entry for more information about its functionality. A second evaluation substitutes the limits of integration into the variable of integration, completing the procedure.

Evaluating \( \int \) in Numerical Results mode (flag \(-3\) set) returns a numerical approximation, and stores the error of integration in variable \( IERR \). \( \int \) consults the number format setting to determine how accurately to compute the result.

**Examples:** In Symbolic Results mode (flag \(-3\) clear) this command sequence:

\[
1 2 '10*x' 'x' \int
\]

returns

\[
'10*(x^2/2) | (x=2) - (10*(x^2/2) | (x=1))'
\]

Subsequent evaluation substitutes the limits of integration, and returns 15.
In Numeric Results mode (flag -3 set) the above command sequence returns the numerical approximation 1. In addition, the variable IERR is created, and contains the error of integration .0000000015.

Related Commands: TAYLR, ∂, Σ

Derivative Function: Takes the derivative of an expression, number, or unit object with respect to a specified variable of differentiation.

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'symb₁'</td>
<td>'name'</td>
<td>→</td>
<td>'symb₂'</td>
</tr>
<tr>
<td>z</td>
<td>'name'</td>
<td>→</td>
<td>0</td>
</tr>
<tr>
<td>x_unit</td>
<td>'name'</td>
<td>→</td>
<td>0</td>
</tr>
</tbody>
</table>

Keyboard Access: ▼▼∂

Affected by Flags: Numerical Results (-3)

Remarks: When executed in stack syntax, ∂ executes a complete differentiation: the expression 'symb₁' is evaluated repeatedly until it contains no derivatives. As part of this process, if the variable of differentiation name has a value, the final form of the expression substitutes that value substituted for all occurrences of the variable. The algebraic syntax for ∂ is '∂name\langle symb₁ \rangle'. When executed in algebraic syntax, ∂ executes a stepwise differentiation of symb₁, invoking the chain rule of differentiation—the result of one evaluation of the expression is the derivative of the argument expression symb₁, multiplied by a new subexpression representing the derivative of symb₁'s argument.

If ∂ is applied to a function for which the HP 48 does not provide a derivative, ∂ returns a new function whose name is der followed by the original function name.
Example: In Radians mode, the command sequence 'dY(SIN(Y))' \(\text{EVAL}\) returns '\(\cos(Y)\times dY(Y)\)'.

When \(Y\) has the value \(X^2\), the command sequence '\(\sin(Y)\times X\ dY\)' \(\text{EVAL}\) returns '\(\cos(X^2)\times (2\times X)\)'. The differentiation has been executed in stack syntax, so that all of the steps of differentiation have been carried out in a single operation.

Related Commands: TAYLR, \(\int\), \(\sum\)

\[
% \text{ Percent Function: Returns } x \text{ (level 2) percent of } y \text{ (level 1).}
\]

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>(\rightarrow)</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>(y)</td>
<td>(\rightarrow)</td>
<td>(xy/100)</td>
</tr>
<tr>
<td>(x)</td>
<td>'symb'</td>
<td>(\rightarrow)</td>
<td>'%(x,symb)'</td>
</tr>
<tr>
<td>'symb'</td>
<td>(x)</td>
<td>(\rightarrow)</td>
<td>'%(symb,x)'</td>
</tr>
<tr>
<td>'symb_1'</td>
<td>'symb_2'</td>
<td>(\rightarrow)</td>
<td>'%(symb_1,symb_2)'</td>
</tr>
<tr>
<td>(x)</td>
<td>(y_unit)</td>
<td>(\rightarrow)</td>
<td>((xy/100)_unit)</td>
</tr>
<tr>
<td>(x_unit)</td>
<td>(y)</td>
<td>(\rightarrow)</td>
<td>((xy/100)_unit)</td>
</tr>
<tr>
<td>'symb'</td>
<td>(x_unit)</td>
<td>(\rightarrow)</td>
<td>'%(symb,x_unit)'</td>
</tr>
<tr>
<td>(x_unit)</td>
<td>'symb'</td>
<td>(\rightarrow)</td>
<td>'%(x_unit,symb)'</td>
</tr>
</tbody>
</table>

Keyboard Access: \(\text{MTH} \text{ REAL} \text{ } \%\%\%\%

Affected by Flags: Numerical Results (-3)

Remarks: Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a difference in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.
The arithmetic operators +, −, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, −, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

For more information on using temperature units with arithmetic functions, see the entry for +.

Example: 23.7 995 % returns 235.815.
15 176_kg % returns 26.4_kg.
100_°C 50 % returns 50_°C.

Related Commands: %CH, %T

\[ \pi \]

\[ \pi \text{ Function:} \quad \text{Returns the symbolic constant } \pi \text{ or its numerical representation, } 3.14159265359. \]

<table>
<thead>
<tr>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>→</td>
<td>' \pi '</td>
</tr>
<tr>
<td></td>
<td>→</td>
<td>3.14159265359</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{\textasciitilde} \pi \)

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

Evaluating \( \pi \) returns its numerical representation if flag −2 or −3 is set; otherwise, returns its symbolic representation.

Remarks: The number returned for \( \pi \) is the closest approximation of the constant \( \pi \) to 12-digit accuracy.

In Radians mode with flags −2 and −3 clear (to return symbolic results), trigonometric functions of \( \pi \) and \( \pi/2 \) are automatically simplified. For example, evaluating 'SIN(\( \pi \))' returns zero. However, if flag −2 or flag −3 is set (to return numerical results),
then evaluating \( \sin(\pi) \) returns the numerical approximation \(-2.06761537357\times10^{-13}\).

**Related Commands:** \( e, i, \text{MAXR}, \text{MINR}, -\pi \)

---

\( \sum \)

**Summation Function:** Calculates the value of a finite series.

<table>
<thead>
<tr>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'indx'</td>
<td>( x_{\text{init}} )</td>
<td>( x_{\text{final}} )</td>
<td>( \text{sumnd} )</td>
<td>( x_{\sum} )</td>
</tr>
<tr>
<td>'indx'</td>
<td>'init'</td>
<td>( x_{\text{final}} )</td>
<td>( \text{sumnd} )</td>
<td>( \Sigma(\text{indx}=\text{init},x_{\text{final}},\text{sumnd})' )</td>
</tr>
<tr>
<td>'indx'</td>
<td>( x_{\text{init}} )</td>
<td>'final'</td>
<td>( \text{sumnd} )</td>
<td>( \Sigma(\text{indx}=x_{\text{init}},\text{final},\text{sumnd})' )</td>
</tr>
<tr>
<td>'indx'</td>
<td>'init'</td>
<td>'final'</td>
<td>( \text{sumnd} )</td>
<td>( \Sigma(\text{indx}=\text{init},\text{final},\text{sumnd})' )</td>
</tr>
</tbody>
</table>

**Keyboard Access:** \( \Rightarrow \Sigma \)

**Affected by Flags:** Symbolic Constants \((-2\)), Numerical Results \((-3)\)

**Remarks:** The summand argument \( \text{sumnd} \) can be a real number, a complex number, a unit object, a local or global name, or an algebraic object.

The algebraic syntax for \( \Sigma \) differs from the stack syntax. The algebraic syntax is this:

\[ \Sigma(\text{index}=\text{initial},\text{final},\text{summand})' \]

**Examples:** In Symbolic Results mode (flags \(-2\) and \(-3\) clear), the command sequence \( 'H' 1 5 'A^N' \) \( \Sigma \) returns \( 'A+A^2+A^3+A^4+A^5' \).

The command sequence \( 'N' 1 'M' 'A^N' \) \( \Sigma \) returns \( '\Sigma(\text{N}=1,\text{M},A^N)' \).

**Related Commands:** TAYLR, \( \int \), \( \partial \)
Sigma Plus Command: Adds one or more data points to the current statistics matrix (reserved variable $\Sigma DAT$).

<table>
<thead>
<tr>
<th>Level m ... Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$\rightarrow$</td>
<td>$x_1 \ldots x_{m-1}$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$[ x_1 \ x_2 \ldots \ x_m ]$</td>
<td>$\rightarrow$</td>
<td>$[ [ x_{11} \ldots x_{1m} ] \ [ x_{n1} \ldots x_{nm} ] ]$</td>
<td>$\rightarrow$</td>
</tr>
</tbody>
</table>

Keyboard Access: $(\rightarrow)^{(STAT)}\ DATA\ \Sigma+$

Affected by Flags: None

Remarks: For a statistics matrix with $m$ columns, arguments for $\Sigma+$ can be entered several ways:

- To enter one data point with a single coordinate value, the argument for $\Sigma+$ must be a real number.
- To enter one data point with multiple coordinate values, the argument for $\Sigma+$ must be a vector of $m$ real coordinate values.
- To enter several data points, the argument for $\Sigma+$ must be a matrix of $n$ rows of $m$ real coordinate values.

In each case, the coordinate values of the data point(s) are added as new rows to the current statistics matrix (reserved variable $\Sigma DAT$). If $\Sigma DAT$ does not exist, $\Sigma+$ creates an $n \times m$ matrix and stores the matrix in $\Sigma DAT$. If $\Sigma DAT$ does exist, an error occurs if it does not contain a real matrix, or if the number of coordinate values in each data point entered with $\Sigma+$ does not match the number of columns in the current statistics matrix.

Once $\Sigma DAT$ exists, individual data points of $m$ coordinates can be entered as $m$ separate real numbers or an $m$-element vector.

LASTARG returns the $m$-element vector in either case.

Example: The sequence $\text{CLZ}[2 \ 3 \ 4]\ \Sigma+\ 3 \ 1 \ 7\ \Sigma+$ creates the matrix $[[2 \ 3 \ 4] \ [3 \ 1 \ 7]]$ in $\Sigma DAT$. **Command Reference 3-419**
Σ+

Related Commands: CLΣ, RCLΣ, STOΣ, Σ−

Σ−

Sigma Minus Command: Returns a vector of \( m \) real numbers (or one number \( x \) if \( m = 1 \)) corresponding to the coordinate values of the last data point entered by Σ+ into the current statistics matrix (reserved variable \( ΣDAT \)).

<table>
<thead>
<tr>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>( x )</td>
</tr>
<tr>
<td>→</td>
<td>( [ x_1 \ x_2 \ ... \ x_m ] )</td>
</tr>
</tbody>
</table>

Keyboard Access: \( \text{←STAT} \ Σ− \)

Affected by Flags: None

Remarks: The last row of the statistics matrix is deleted.

The vector returned by \( Σ− \) can be edited or replaced, then restored to the statistics matrix by Σ+.

Related Commands: CLΣ, RCLΣ, STOΣ, Σ+

√

Square Root Analytic Function: Returns the (positive) square root of the argument.
For a complex number \((x_1, y_1)\), the square root is this complex number:

\[
(x_2, y_2) = (\sqrt{r} \cos \frac{\theta}{2}, \sqrt{r} \sin \frac{\theta}{2})
\]

where \(r = \text{ABS}(x_1, y_1)\), and \(\theta = \text{ARG}(x_1, y_1)\).

If \((x_1, y_1) = (0, 0)\), then the square root is \((0, 0)\).

The inverse of SQ is a relation, not a function, since SQ sends more than one argument to the same result. The inverse relation for SQ is expressed by ISOL as this general solution:

\[1 \in 1*\text{SQ}\]

The function \(\sqrt{}\) is the inverse of a part of SQ, a part defined by restricting the domain of SQ such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of SQ are called the principal values of the inverse relation. The \(\sqrt{}\) function in its entirety is called the principal branch of the inverse relation, and the points sent by \(\sqrt{}\) to the boundary of the restricted domain of SQ form the branch cuts of \(\sqrt{}\).

The principal branch used by the HP 48 for \(\sqrt{}\) was chosen because it is analytic in the regions where the arguments of the real-valued inverse function are defined. The branch cut for the complex-valued square root function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of \(\sqrt{}\). The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of
the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation \( \sqrt{1 \pm \frac{1}{2}} \) for the case \( sI = 1 \).
For the other value of \( sI \), the half-plane in the lower graph is rotated.
Taken together, the half-planes cover the whole complex plane, which is the domain of \( SQ \).

View these graphs with domain and range reversed to see how the domain of \( SQ \) is restricted to make an inverse function possible.
Consider the half-plane in the lower graph as the restricted domain \( Z = \langle x, y \rangle \). \( SQ \) sends this domain onto the whole complex plane in the range \( W = \langle u, v \rangle = SQ \langle x, y \rangle \) in the upper graph.

**Related Commands:** \( SQ, ^\wedge, ISOL \)

---

**Diagram:**

- **Domain:** \( Z = (x,y) \)
- **Range:** \( W = (u,v) = \sqrt{(x,y)} \)
### Attach Unit Function

**Definition:** Attaches a unit expression to a real number. Performed automatically in the Unit Catalog (`Units`).

**Keyboard Access:**

**Affected by Flags:** None

**Related Commands:** →UNIT

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>'unit expression'</td>
<td>→</td>
<td><code>x_unit</code></td>
</tr>
</tbody>
</table>

### Where Function

**Definition:** Substitutes values for names in an expression.

**Keyboard Access:**

**Affected by Flags:** Numerical Results (−3)

**Remarks:** is used primarily in algebraic objects, where its syntax is:

```plaintext
'symb_{old}' \{ name_1 \ 'symb_1' \ name_2 \ 'symb_2' \ ... \} \rightarrow \ 'symb_{new}'  

x \{ name_1 \ 'symb_1' \ name_2 \ 'symb_2' \ ... \} \rightarrow \ x  

(x, y) \{ name_1 \ 'symb_1' \ name_2 \ 'symb_2' \ ... \} \rightarrow \ (x, y)
```

It enables algebraics to include variable-like substitution information about names. Symbolic functions that delay name evaluation (such as ∫ and ∂) can then extract substitution information from local...
variables and include that information in the expression, avoiding the problem that would occur if the local variables no longer existed when the local names were finally evaluated.

**Related Commands:** APPLY, QUOTE

---

Create Local Variables Command: Creates local variables.

<table>
<thead>
<tr>
<th>Level n ... Level 1</th>
<th>→</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>obj_1 ... obj_n</code></td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

**Keyboard Access:** (→)<br>
**Affected by Flags:** None

**Remarks:** *Local variable structures* specify one or more local variables and a defining procedure.

A local variable structure consists of the → command, followed by one or more names, followed by a defining procedure—either a program or an algebraic. The → command stores objects from the stack into local variables with the specified names. The resultant local variables exist only while the defining procedure is being executed. The syntax of a local variable structure is one of the following:

- `name_1 name_2 ... name_n ← program`<br>
- `name_1 name_2 ... name_n 'algebraic expression'

**Example:** This program:

```
← × y ← × y × y − + ↑
```

takes an object from level 2 and stores it in local variable `x`, takes an object from level 1 and stores it in local variable `y`, and executes calculations with `x` and `y` in the defining procedure (in this case a program). When the defining procedure ends, local variables `x` and `y` disappear.
User-Defined Functions. A user-defined function is a variable containing a program that consists solely of a local variable structure. For example, the variable $A$, containing this program:

$$A \to x \ y \ z \ 'x*y/2+z'$$

is a user-defined function. Like a built-in function, a user-defined function can take its arguments in stack syntax or algebraic syntax, and can take symbolic arguments. In addition, a user-defined function is differentiable if its defining procedure is an algebraic expression that contains only differentiable functions.

Related Commands: DEFINE, STO
The Equation Library consists of 15 subjects and more than 100 titles. Each subject and title has a number that you can use with SOLVEQN to specify the set of equations. These numbers are shown in parentheses after the headings.

See the end of this section for references given in each subject (Reference: x). Remember that some equations are estimates and assume certain conditions. See the references or other standard texts for assumptions and limitations of the equations.

Solutions in the examples have been rounded to four decimal places.

---

**Columns and Beams (1)**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e$</td>
<td>Eccentricity (offset) of load</td>
</tr>
<tr>
<td>$\sigma_{cr}$</td>
<td>Critical stress</td>
</tr>
<tr>
<td>$\sigma_{max}$</td>
<td>Maximum stress</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Slope at $x$</td>
</tr>
<tr>
<td>$A$</td>
<td>Cross-sectional area</td>
</tr>
<tr>
<td>$a$</td>
<td>Distance to point load</td>
</tr>
<tr>
<td>$c$</td>
<td>Distance to edge fiber (Eccentric Columns), or Distance to applied moment (beams)</td>
</tr>
<tr>
<td>$E$</td>
<td>Modulus of elasticity</td>
</tr>
<tr>
<td>$I$</td>
<td>Moment of inertia</td>
</tr>
</tbody>
</table>
### Variable Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>Effective length factor of column</td>
</tr>
<tr>
<td>$L$</td>
<td>Length of column or beam</td>
</tr>
<tr>
<td>$M$</td>
<td>Applied moment</td>
</tr>
<tr>
<td>$M_x$</td>
<td>Internal bending moment at $x$</td>
</tr>
<tr>
<td>$P$</td>
<td>Load (Eccentric Columns), or Point load (beams)</td>
</tr>
<tr>
<td>$P_{cr}$</td>
<td>Critical load</td>
</tr>
<tr>
<td>$r$</td>
<td>Radius of gyration</td>
</tr>
<tr>
<td>$V$</td>
<td>Shear force at $x$</td>
</tr>
<tr>
<td>$w$</td>
<td>Distributed load</td>
</tr>
<tr>
<td>$x$</td>
<td>Distance along beam</td>
</tr>
<tr>
<td>$y$</td>
<td>Deflection at $x$</td>
</tr>
</tbody>
</table>

For simply supported beams and cantilever beams ("Simple Deflection" through "Cantilever Shear"), the calculations differ depending upon the location of $x$ relative to the loads.

- Applied loads are positive downward.
- The applied moment is positive counterclockwise.
- Deflection is positive upward.
- Slope is positive counterclockwise.
- Internal bending moment is positive counterclockwise on the left-hand part.
- Shear force is positive downward on the left-hand part.

Reference: 2.
Elastic Buckling (1, 1)

These equations apply to a slender column \((K \cdot L/r > 100)\) with length factor \(K\).

\[
P_{cr} = \frac{\pi^2 \cdot E \cdot A}{\left(\frac{K \cdot L}{r}\right)^2}
\]

\[
P_{cr} = \frac{\pi^2 \cdot E \cdot I}{(K \cdot L)^2}
\]

\[
\sigma_{cr} = \frac{P_{cr}}{A}
\]

\[
r = \sqrt{\frac{I}{A}}
\]

Example:

Given: \(L=7.3152 \text{ m}, r=4.1148 \text{ cm}, E=199947961.502 \text{ kPa}, A=53.0967 \text{ cm}^2, K=0.7, I=8990598.7930 \text{ mm}^4\).

Solution: \(P_{cr}=676.6019 \text{ kN}, \sigma_{cr}=127428.2444 \text{ kPa}\).

Eccentric Columns (1, 2)

See “Elastic Buckling.”
Equations:

\[
\sigma_{\text{max}} = \frac{P}{A} \left(1 + \frac{\epsilon \cdot c}{r^2} \cdot \left(\frac{1}{\cos \left(\frac{K \cdot L}{2 \cdot r} \cdot \sqrt{\frac{P}{E \cdot A}}\right)}\right)\right)
\]

\[
r = \sqrt{\frac{I}{A}}
\]

Example:

Given: \(L=6.6542\text{ m}, A=187.9351\text{ cm}^2, r=8.4836\text{ cm}, E=206842718.795\text{ kPa}, I=135259652.16\text{ mm}^4, K=1, P=1908.2571\text{ kN}, \epsilon=15.24\text{ cm}, c=1.1806\text{ cm.}\)

Solution: \(\sigma_{\text{max}}=140853.0970\text{ kPa.}\)

Simple Deflection (1, 3)

Equation:

\[
y = \frac{P \cdot \left(L - a\right) \cdot x}{6 \cdot L \cdot E \cdot I} \cdot \left(x^2 + \left(L - a\right)^2 - L^2\right)
- \frac{M \cdot x}{E \cdot I} \cdot \left(c - \frac{x^2}{6 \cdot L} - \frac{L}{3} - \frac{c^2}{2 \cdot L}\right)
- \frac{w \cdot x}{24 \cdot E \cdot I} \cdot \left(L^3 + x^2 \cdot \left(x - 2 \cdot L\right)\right)
\]
Example:

**Given:** $L=20\,\text{ft}$, $E=29000000\,\text{psi}$, $I=40\,\text{in}^4$, $a=10\,\text{ft}$, 
$P=674.427\,\text{lbf}$, $c=17\,\text{ft}$, $M=3687.81\,\text{ft}\cdot\text{lbf}$, $w=102.783\,\text{lbf/ft}$, 
$x=9\,\text{ft}$.

**Solution:** $y=-.6005\,\text{in}$.

**Simple Slope (1, 4)**

![Simple Slope Diagram](image)

**Equation:**

$$
\Theta = \frac{P \cdot (L - a)}{6 \cdot L \cdot E \cdot I} \cdot \left( 3 \cdot x^2 + \left( L - a \right)^2 - L^2 \right) \\
- \frac{M}{E \cdot I} \cdot \left( c - \frac{x^2}{2 \cdot L} - \frac{L}{3} - \frac{c^2}{2 \cdot L} \right) \\
- \frac{w}{24 \cdot E \cdot I} \cdot \left( L^3 + x^2 \cdot \left( 4 \cdot x - 6 \cdot L \right) \right)
$$

**Example:**

**Given:** $L=20\,\text{ft}$, $E=29000000\,\text{psi}$, $I=40\,\text{in}^4$, $a=10\,\text{ft}$, 
$P=674.427\,\text{lbf}$, $c=17\,\text{ft}$, $M=3687.81\,\text{ft}\cdot\text{lbf}$, $w=102.783\,\text{lbf/ft}$, 
$x=9\,\text{ft}$.

**Solution:** $\theta=-.0876\,^\circ$. 

**Equation Reference** 4-5
Simple Moment (1, 5)

Equation:

\[ M_x = \frac{P \cdot \left( L - a \right)}{L} \cdot x + \frac{M \cdot x}{L} + \frac{w \cdot x}{2} \cdot \left( L - x \right) \]

Example:

Given: \( L=20\text{ ft}, a=10\text{ ft}, P=674.427\text{ lbf}, c=17\text{ ft}, M=3687.81\text{ ft-lbf}, w=102.783\text{ lbf/ft}, x=9\text{ ft}. \)

Solution: \( M_x=9782.1945\text{ ft-lbf}. \)

Simple Shear (1, 6)

Equation:

\[ V = \frac{P \cdot \left( L - a \right)}{L} + \frac{M}{L} + \frac{w}{2} \cdot \left( L - 2 \cdot x \right) \]
Example:

Given: \( L=20 \text{ ft}, a=10 \text{ ft}, P=674.427 \text{ lbf}, M=3687.81 \text{ ft-lbf}, w=102.783 \text{ lbf/ft}, x=9 \text{ ft} \).

Solution: \( V=624.387 \text{ lbf} \).

Cantilever Deflection (1, 7)

Equation:

\[
y = \frac{P \cdot x^2}{6 \cdot E \cdot I} \left( x - 3 \cdot a \right) + \frac{M \cdot x^2}{2 \cdot E \cdot I} - \frac{w \cdot x^2}{24 \cdot E \cdot I} \left( 6 \cdot L^2 - 4 \cdot L \cdot x + x^2 \right)
\]

Example:

Given: \( L=10 \text{ ft}, E=29000000 \text{ psi}, I=15 \text{ in}^4, P=500 \text{ lbf}, M=800 \text{ ft-lbf}, a=3 \text{ ft}, c=6 \text{ ft}, w=100 \text{ lbf/ft}, x=8 \text{ ft} \).

Solution: \( y=-.3316 \text{ in} \).

Cantilever Slope (1, 8)
Equation:

\[ \Theta = \frac{P \cdot x}{2 \cdot E \cdot I} \left( x - 2 \cdot a \right) + \frac{M \cdot x}{E \cdot I} - \frac{w \cdot x}{6 \cdot E \cdot I} \left( 3 \cdot L^2 - 3 \cdot L \cdot x + x^2 \right) \]

Example:
Given: \( L=10\, \text{ft}, \ E=29000000\, \text{psi}, \ I=15\, \text{in}^4, \ P=500\, \text{lbf}, \ M=800\, \text{ft} \cdot \text{lbf}, \ a=3\, \text{ft}, \ c=6\, \text{ft}, \ w=100\, \text{lbf/ft}, \ x=8\, \text{ft} \).
Solution: \( \Theta=-.2652^\circ \).

Cantilever Moment (1, 9)

Equation:

\[ M_x = P \cdot \left( x - a \right) + M - \frac{W}{2} \left( L^2 - 2 \cdot L \cdot x + x^2 \right) \]

Example:
Given: \( L=10\, \text{ft}, \ P=500\, \text{lbf}, \ M=800\, \text{ft} \cdot \text{lbf}, \ a=3\, \text{ft}, \ c=6\, \text{ft}, \ w=100\, \text{lbf/ft}, \ x=8\, \text{ft} \).
Solution: \( M_x=-200\, \text{ft} \cdot \text{lbf} \).
Cantilever Shear (1, 10)

Equation:

\[ V = P + w \cdot (L - x) \]

Example:

Given: \( L = 10\text{ ft}, P = 500\text{ lbf}, a = 3\text{ ft}, x = 8\text{ ft}, w = 100\text{ lbf/ft}. \)

Solution: \( V = 200\text{ lbf}. \)

Electricity (2)

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon_r )</td>
<td>Relative permittivity</td>
</tr>
<tr>
<td>( \mu_r )</td>
<td>Relative permeability</td>
</tr>
<tr>
<td>( \omega )</td>
<td>Angular frequency</td>
</tr>
<tr>
<td>( \omega_0 )</td>
<td>Resonant angular frequency</td>
</tr>
<tr>
<td>( \phi )</td>
<td>Phase angle</td>
</tr>
<tr>
<td>( \phi_p, \phi_s )</td>
<td>Parallel and series phase angles</td>
</tr>
<tr>
<td>( \rho )</td>
<td>Resistivity</td>
</tr>
</tbody>
</table>
### Variable Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta I$</td>
<td>Current change</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>Time change</td>
</tr>
<tr>
<td>$\Delta V$</td>
<td>Voltage change</td>
</tr>
<tr>
<td>$A$</td>
<td>Wire cross-section area (Wire Resistance), or Solenoid cross-section area (Solenoid Inductance), or Plate area (Plate Capacitor)</td>
</tr>
<tr>
<td>$C,C_1,C_2$</td>
<td>Capacitance</td>
</tr>
<tr>
<td>$C_p,C_s$</td>
<td>Parallel and series capacitances</td>
</tr>
<tr>
<td>$d$</td>
<td>Plate separation</td>
</tr>
<tr>
<td>$E$</td>
<td>Energy</td>
</tr>
<tr>
<td>$F$</td>
<td>Force between charges</td>
</tr>
<tr>
<td>$f$</td>
<td>Frequency</td>
</tr>
<tr>
<td>$f_0$</td>
<td>Resonant frequency</td>
</tr>
<tr>
<td>$I$</td>
<td>Current, or Total current (Current Divider)</td>
</tr>
<tr>
<td>$I_1$</td>
<td>Current in $R_1$</td>
</tr>
<tr>
<td>$I_{max}$</td>
<td>Maximum current</td>
</tr>
<tr>
<td>$L$</td>
<td>Inductance, or Length (Wire Resistance, Cylindrical Capacitor)</td>
</tr>
<tr>
<td>$L_1,L_2$</td>
<td>Inductance</td>
</tr>
<tr>
<td>$L_p,L_s$</td>
<td>Parallel and series inductances</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of turns</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of turns per unit length</td>
</tr>
<tr>
<td>$P$</td>
<td>Power</td>
</tr>
<tr>
<td>$q$</td>
<td>Charge</td>
</tr>
<tr>
<td>$q_1,q_2$</td>
<td>Point charge</td>
</tr>
<tr>
<td>$Q_p,Q_s$</td>
<td>Parallel and series quality factors</td>
</tr>
<tr>
<td>$r$</td>
<td>Charge distance</td>
</tr>
<tr>
<td>$R,R_1,R_2$</td>
<td>Resistance</td>
</tr>
<tr>
<td>$r_i,r_o$</td>
<td>Inside and outside radii</td>
</tr>
<tr>
<td>$R_p,R_s$</td>
<td>Parallel and series resistances</td>
</tr>
</tbody>
</table>
Variable Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>Time</td>
</tr>
<tr>
<td>$t_i, t_f$</td>
<td>Initial and final times</td>
</tr>
<tr>
<td>$V$</td>
<td>Voltage, or Total voltage (Voltage Divider)</td>
</tr>
<tr>
<td>$V_1$</td>
<td>Voltage across R1</td>
</tr>
<tr>
<td>$V_i, V_f$</td>
<td>Initial and final voltages</td>
</tr>
<tr>
<td>$V_{max}$</td>
<td>Maximum voltage</td>
</tr>
<tr>
<td>$X_C$</td>
<td>Reactance of capacitor</td>
</tr>
<tr>
<td>$X_L$</td>
<td>Reactance of inductor</td>
</tr>
</tbody>
</table>

Reference: 3.

**Coulomb’s Law (2, 1)**

This equation describes the electrostatic force between two charged particles.

**Equation:**

$$ F = \frac{1}{4 \pi \varepsilon_0 \varepsilon_r} \cdot \left( \frac{q_1 \cdot q_2}{r^2} \right) $$

**Example:**

*Given:* $q_1 = 1.6 \times 10^{-19} \text{C}$, $q_2 = 1.6 \times 10^{-19} \text{C}$, $r = 4.00 \times 10^{-13} \text{cm}$, $\varepsilon_r = 1.00$.

*Solution:* $F = 14.3801 \text{N}$.

**Ohm’s Law and Power (2, 2)**

**Equations:**

$$ V = I \cdot R \quad P = V \cdot I \quad P = I^2 \cdot R \quad P = \frac{V^2}{R} $$

Equation Reference 4-11
Example:

Given: $V=24\_V$, $I=16\_A$.

Solution: $R=1.5\_\Omega$, $P=384\_W$.

**Voltage Divider (2, 3)**

![Voltage Divider Diagram]

Equation:

$$V_1 = V \cdot \left( \frac{R_1}{R_1 + R_2} \right)$$

Example:

Given: $R_1=40\_\Omega$, $R_2=10\_\Omega$, $V=100\_V$.

Solution: $V_1=80\_V$.

**Current Divider (2, 4)**

![Current Divider Diagram]
Equation:

\[ I_1 = I \cdot \left( \frac{R_2}{R_1 + R_2} \right) \]

Example:

Given: \( R_1 = 10 \, \Omega \), \( R_2 = 6 \, \Omega \), \( I = 15 \, \text{A} \).

Solution: \( I_1 = 5.6250 \, \text{A} \).

**Wire Resistance (2, 5)**

Equation:

\[ R = \frac{\rho \cdot L}{A} \]

Example:

Given: \( \rho = 0.0035 \, \Omega \cdot \text{cm} \), \( L = 50 \, \text{cm} \), \( A = 1 \, \text{cm}^2 \).

Solution: \( R = 0.175 \, \Omega \).

**Series and Parallel R (2, 6)**

![Diagram of series and parallel resistances]

Equations:

\[ R_s = R_1 + R_2 \]

\[ \frac{1}{R_p} = \frac{1}{R_1} + \frac{1}{R_2} \]
Example:
Given: \( R1=2\ \Omega, \) \( R2=3\ \Omega. \)
Solution: \( Rs=5\ \Omega, \) \( Rp=1.2000\ \Omega. \)

Series and Parallel C (2, 7)

\[ \frac{1}{C_s} = \frac{1}{C_1} + \frac{1}{C_2} \]
\[ C_p = C_1 + C_2 \]

Example:
Given: \( C1=2\ \mu F, \) \( C2=3\ \mu F. \)
Solution: \( C_s=1.2000\ \mu F, \) \( C_p=5\ \mu F. \)

Series and Parallel L (2, 8)
Equations:

\[ L_s = L_1 + L_2 \]
\[ \frac{1}{L_p} = \frac{1}{L_1} + \frac{1}{L_2} \]

Example:

Given: \( L_1 = 17 \text{ mH}, L_2 = 16.5 \text{ mH} \).
Solution: \( L_s = 33.5000 \text{ mH}, L_p = 8.3731 \text{ mH} \).

Capacitive Energy (2, 9)

Equation:

\[ E = \frac{C \cdot V^2}{2} \]

Example:

Given: \( E = 0.25 \text{ J}, C = 20 \mu \text{F} \).
Solution: \( V = 50 \text{ V} \).

Inductive Energy (2, 10)

Equation:

\[ E = \frac{L \cdot I^2}{2} \]

Example:

Given: \( E = 4 \text{ J}, L = 15 \text{ mH} \).
Solution: \( I = 23.0940 \text{ A} \).
RLC Current Delay (2, 11)

The phase delay (angle) is positive for current lagging voltage.

Equations:

\[\tan(\phi_s) = \frac{XL - XC}{R}\]
\[\tan(\phi_p) = \frac{1}{\frac{1}{XC} - \frac{1}{XL}}\]
\[XC = \frac{1}{\omega \cdot C}\]
\[XL = \omega \cdot L\]
\[\omega = 2 \cdot \pi \cdot f\]

Example:

Given: \(f=107\) Hz, \(C=80\) \(\mu\)F, \(L=20\) mH, \(R=5\) \(\Omega\).

Solution: \(\omega=672.3008\) \(r/s\), \(\phi_s=-45.8292^\circ\), \(\phi_p=-5.8772^\circ\), \(XC=18.5929\) \(\Omega\), \(XL=13.4460\) \(\Omega\).

DC Capacitor Current (2, 12)

These equations approximate the dc current required to change the voltage on a capacitor in a certain time interval.

Equations:

\[I = C \cdot \left(\frac{\Delta V}{\Delta t}\right)\]
\[\Delta V = V_f - V_i\]
\[\Delta t = t_f - t_i\]
Example:
Given: $C=15\,\mu F$, $V_i=2.3\,V$, $V_f=3.2\,V$, $I=10\,A$, $t_i=0\,s$.
Solution: $\Delta V=0.9000\,V$, $\Delta t=1.3500\,\mu s$, $t_f=1.3500\,\mu s$.

Capacitor Charge (2, 13)

Equation:
\[
q = CV.
\]

Example:
Given: $C'=20\,\mu F$, $V=100\,V$.
Solution: $q=0.0020\,C$.

DC Inductor Voltage (2, 14)

These equations approximate the dc voltage induced in an inductor by a change in current in a certain time interval.

Equations:
\[
V = -L \cdot \left( \frac{\Delta I}{\Delta t} \right), \quad \Delta I = I_f - I_i, \quad \Delta t = t_f - t_i
\]

Example:
Given: $L=100\,mH$, $V=52\,V$, $\Delta t=32\,\mu s$, $I_i=23\,A$, $t_i=0\,s$.
Solution: $\Delta I=-0.0166\,A$, $I_f=22.9834\,A$, $t_f=32\,\mu s$. 

Equation Reference 4-17
**RC Transient (2, 15)**

Equation:

\[ V = V_f - \left( V_f - V_i \right) e^{-\frac{t}{R C}} \]

Example:

Given: \( V_i = 0 \, V \), \( C = 50 \, \mu F \), \( V_f = 10 \, V \), \( R = 100 \, \Omega \), \( t = 2 \, ms \).

Solution: \( V = 3.2968 \, V \).

**RL Transient (2, 16)**

Equation:

\[ I = \frac{1}{R} \left( V_f - \left( V_f - V_i \right) e^{-\frac{t \cdot R}{L}} \right) \]
Example:
Given: $V_i=0\_V, \ V_f=5\_V, \ R=50\_\omega, \ L=50\_mH, \ t=75\_\mu s$.
Solution: $I=0.0072\_A$.

Resonant Frequency (2, 17)

Equations:

$$\omega_0 = \frac{1}{\sqrt{L \cdot C}} \quad Q_s = \frac{1}{R} \cdot \sqrt{\frac{L}{C}} \quad Q_p = R \cdot \sqrt{\frac{C}{L}} \quad \omega_0 = 2 \cdot \pi \cdot f_0$$

Example:
Given: $L=500\_mH, \ C=8\_\mu F, \ R=10\_\omega$.
Solution: $\omega_0=500\_r/s, \ Q_s=25.0000, \ Q_p=0.0400, \ f_0=79.5775\_Hz$.

Plate Capacitor (2, 18)

Equation:

$$C = \frac{\epsilon_0 \cdot \epsilon_r \cdot A}{d}$$

Example:
Given: $C=25\_\mu F, \ \epsilon_r=2.26, \ A=1\_cm^2$.
Solution: $d=8.0042E-9\_cm$. 

Equation Reference 4-19
Cylindrical Capacitor (2, 19)

Equation:

\[ C = \frac{2 \cdot \pi \cdot \varepsilon_0 \cdot \varepsilon_r \cdot L}{\ln \left( \frac{r_o}{r_i} \right)} \]

Example:

Given: \( \varepsilon_r = 1 \), \( r_o = 1 \text{ cm} \), \( r_i = 0.999 \text{ cm} \), \( L = 10 \text{ cm} \).

Solution: \( C = 0.0056 \mu \text{F} \).

Solenoid Inductance (2, 20)

Equation:

\[ L = \mu_0 \cdot \mu_r \cdot n^2 \cdot A \cdot h \]

Example:

Given: \( \mu_r = 2.5 \), \( n = 40 \text{ /cm} \), \( A = 0.2 \text{ cm}^2 \), \( h = 3 \text{ cm} \).

Solution: \( L = 0.0302 \text{ mH} \).
Toroid Inductance (2, 21)

Equation:
\[ L = \frac{\mu_0 \cdot \mu_r \cdot N^2 \cdot h}{2 \cdot \pi} \cdot \ln \left( \frac{r_o}{r_i} \right) \]

Example:
Given: \( \mu_r = 1, N = 5000, h = 2\, \text{cm}, r_i = 2\, \text{cm}, r_o = 4\, \text{cm} \).
Solution: \( L = 69.3147\, \text{mH} \).

Sinusoidal Voltage (2, 22)

Equations:
\[ V = V_{max} \cdot \sin \left( \omega \cdot t + \phi \right) \quad \omega = 2 \cdot \pi \cdot f \]

Example:
Given: \( V_{max} = 110\, \text{V}, t = 30\, \mu\text{s}, f = 60\, \text{Hz}, \phi = 15^\circ \).
Solution: \( \omega = 376.9911\, \text{r/s}, V = 29.6699\, \text{V} \).

Sinusoidal Current (2, 23)

Equations:
\[ I = I_{max} \cdot \sin \left( \omega \cdot t + \phi \right) \quad \omega = 2 \cdot \pi \cdot f \]
Example:

Given: $t=32\_s$, $Imax=10\_A$, $\omega=636\_r/s$, $\phi=30\_\circ$.


---

**Fluids (3)**

### Variable Names and Descriptions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>Roughness</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Dynamic viscosity</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Density</td>
</tr>
<tr>
<td>$\Delta P$</td>
<td>Pressure change</td>
</tr>
<tr>
<td>$\Delta y$</td>
<td>Height change</td>
</tr>
<tr>
<td>$\Sigma K$</td>
<td>Total fitting coefficients</td>
</tr>
<tr>
<td>$A$</td>
<td>Cross-sectional area</td>
</tr>
<tr>
<td>$A1,A2$</td>
<td>Initial and final cross-sectional areas</td>
</tr>
<tr>
<td>$D$</td>
<td>Diameter</td>
</tr>
<tr>
<td>$D1,D2$</td>
<td>Initial and final diameters</td>
</tr>
<tr>
<td>$h$</td>
<td>Depth relative to $P0$ reference depth</td>
</tr>
<tr>
<td>$hL$</td>
<td>Head loss</td>
</tr>
<tr>
<td>$L$</td>
<td>Length</td>
</tr>
<tr>
<td>$M$</td>
<td>Mass flow rate</td>
</tr>
<tr>
<td>$n$</td>
<td>Kinematic viscosity</td>
</tr>
<tr>
<td>$P$</td>
<td>Pressure at $h$</td>
</tr>
<tr>
<td>$P0$</td>
<td>Reference pressure</td>
</tr>
<tr>
<td>$P1,P2$</td>
<td>Initial and final pressures</td>
</tr>
<tr>
<td>$Q$</td>
<td>Volume flow rate</td>
</tr>
<tr>
<td>$Re$</td>
<td>Reynolds number</td>
</tr>
<tr>
<td>$v1,v2$</td>
<td>Initial and final velocities</td>
</tr>
<tr>
<td>$vavg$</td>
<td>Average velocity</td>
</tr>
<tr>
<td>$W$</td>
<td>Power input</td>
</tr>
<tr>
<td>$y1,y2$</td>
<td>Initial and final heights</td>
</tr>
</tbody>
</table>

References: 3, 6, 9.
Pressure at Depth (3, 1)

This equation describes hydrostatic pressure for an incompressible fluid. Depth $h$ is positive downward from the reference.

Equation:

$$P = P_0 + \rho \cdot g \cdot h$$

Example:

**Given:** $h=100\text{ m}$, $\rho=1025.1817\text{ kg/m}^3$, $P_0=1\text{ atm}$.

**Solution:** $P=1106.6848\text{ kPa}$.

Bernoulli Equation (3, 2)

These equations represent the streamlined flow of an incompressible fluid.
Equations:

\[
\frac{\Delta P}{\rho} + \frac{v_2^2 - v_1^2}{2} + g \cdot \Delta y = 0
\]

\[
\frac{\Delta P}{\rho} + \frac{v_2^2 \cdot \left( 1 - \left( \frac{A_2}{A_1} \right)^2 \right)}{2} + g \cdot \Delta y = 0
\]

\[
\frac{\Delta P}{\rho} + \frac{v_1^2 \cdot \left( \left( \frac{A_1}{A_2} \right)^2 - 1 \right)}{2} + g \cdot \Delta y = 0
\]

\[
\Delta P = P_2 - P_1 \quad \Delta y = y_2 - y_1 \quad M = \rho \cdot Q
\]

\[
Q = A_2 \cdot v_2 \quad Q = A_1 \cdot v_1
\]

\[
A_1 = \frac{\pi \cdot D_1^2}{4} \quad A_2 = \frac{\pi \cdot D_2^2}{4}
\]

Example:

**Given:** \( P_2 = 25 \text{ psi}, \ P_1 = 75 \text{ psi}, \ y_2 = 35 \text{ ft}, \ y_1 = 0 \text{ ft}, \ D_1 = 18 \text{ in}, \rho = 64 \text{ lb/ft}^3, \ v_1 = 100 \text{ ft/s}. \)

**Solution:** \( Q = 10602.8752 \text{ ft}^3/\text{min}, \ M = 678584.0132 \text{ lb/min}, \ v_2 = 122.4213 \text{ ft/s}, \ A_2 = 207.8633 \text{ in}^2, \ D_2 = 16.2684 \text{ in}, \ A_1 = 254.4690 \text{ in}^2, \ \Delta P = -50 \text{ psi}, \ \Delta y = 35 \text{ ft}. \)

**Flow with Losses (3, 3)**

These equations extend Bernoulli’s equation to include power input (or output) and head loss.
Equations:

\[ M \cdot \left( \frac{\Delta P}{\rho} + \frac{v_2^2 - v_1^2}{2} + g \cdot \Delta y + hL \right) = W \]

\[ M \cdot \left( \frac{\Delta P}{\rho} + \frac{v_2^2 \cdot \left( \frac{1 - \left( \frac{A_2}{A_1} \right)^2}{2} \right)}{2} + g \cdot \Delta y + hL \right) = W \]

\[ \Delta P = P_2 - P_1 \quad \Delta y = y_2 - y_1 \quad M = \rho \cdot Q \]

\[ Q = A_2 \cdot v_2 \quad Q = A_1 \cdot v_1 \]

\[ A_1 = \frac{\pi \cdot D_1^2}{4} \quad A_2 = \frac{\pi \cdot D_2^2}{4} \]

Example:

Given: \( P_2 = 30 \text{ psi}, \ P_1 = 65 \text{ psi}, \ y_2 = 100 \text{ ft}, \ y_1 = 0 \text{ ft}, \ \rho = 64 \text{ lb/ft}^3, \ D_1 = 24 \text{ in}, \ hL = 2.0 \text{ ft}^2/\text{s}^2, \ W = 25 \text{ hp}, \ v_1 = 100 \text{ ft/s}. \)

Solution: \( Q = 18849.5559 \text{ ft}^3/\text{min}, \ M = 1206371.5790 \text{ lb/min}, \ \Delta P = -35 \text{ psi}, \ \Delta y = 100 \text{ ft}, \ v_2 = 93.1269 \text{ ft/s}, \ A_1 = 452.3893 \text{ in}^2, \ A_2 = 485.7773 \text{ in}^2, \ D_2 = 24.8699 \text{ in}. \)
Flow in Full Pipes (3, 4)

These equations adapt Bernoulli’s equation for flow in a round, full pipe, including power input (or output) and frictional losses. (See “FANNING” in chapter 3.)

Equations:

\[ \rho \cdot \left( \frac{\pi \cdot D^2}{4} \right) \cdot \text{vavg} \cdot \left( \frac{\Delta P}{\rho} + g \cdot \Delta y + \text{vavg}^2 \cdot \left( 2 \cdot f \cdot \left( \frac{L}{D} \right) + \frac{\Sigma K}{2} \right) \right) = W \]

\[ \Delta P = P_2 - P_1 \quad \Delta y = y_2 - y_1 \quad M = \rho \cdot Q \]

\[ Q = A \cdot \text{vavg} \quad A = \frac{\pi \cdot D^2}{4} \]

\[ \text{Re} = \frac{D \cdot \text{vavg} \cdot \rho}{\mu} \quad n = \frac{\mu}{\rho} \]

Example:

Given: \( \rho = 62.4 \text{ lb/ft}^3 \), \( D = 12 \text{ in} \), \( \text{vavg} = 8 \text{ ft/s} \), \( P_2 = 15 \text{ psi} \), \( P_1 = 20 \text{ psi} \), \( y_2 = 40 \text{ ft} \), \( y_1 = 0 \text{ ft} \), \( \mu = 0.00002 \text{ lbf*sec/ft}^2 \), \( \Sigma K = 2.25 \), \( \epsilon = 0.02 \text{ in} \), \( L = 250 \text{ ft} \).

Solution: \( \Delta P = -5 \text{ psi} \), \( \Delta y = 40 \text{ ft} \), \( A = 113.0973 \text{ in}^2 \), \( n = 1.0312 \text{ ft}^2/\text{s} \), \( Q = 376.9911 \text{ ft}^3/\text{min} \), \( M = 23524.2458 \text{ lb/ min} \), \( W = 25.8897 \text{ hp} \), \( \text{Re} = 775780.5 \).
## Forces and Energy (4)

### Variable Names and Descriptions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Angular acceleration</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Angular velocity</td>
</tr>
<tr>
<td>$\omega_i, \omega_f$</td>
<td>Initial and final angular velocities</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Fluid density</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Torque</td>
</tr>
<tr>
<td>$\Theta$</td>
<td>Angular displacement</td>
</tr>
<tr>
<td>$a$</td>
<td>Acceleration</td>
</tr>
<tr>
<td>$A$</td>
<td>Projected area relative to flow</td>
</tr>
<tr>
<td>$a_r$</td>
<td>Centripetal acceleration at $r$</td>
</tr>
<tr>
<td>$a_t$</td>
<td>Tangential acceleration at $r$</td>
</tr>
<tr>
<td>$C_d$</td>
<td>Drag coefficient</td>
</tr>
<tr>
<td>$E$</td>
<td>Energy</td>
</tr>
<tr>
<td>$F$</td>
<td>Force at $r$ or $x$, or Spring force (Hooke’s Law), or Attractive force (Law of Gravitation), or Drag force (Drag Force)</td>
</tr>
<tr>
<td>$I$</td>
<td>Moment of inertia</td>
</tr>
<tr>
<td>$k$</td>
<td>Spring constant</td>
</tr>
<tr>
<td>$K_i, K_f$</td>
<td>Initial and final kinetic energies</td>
</tr>
<tr>
<td>$m, m_1, m_2$</td>
<td>Mass</td>
</tr>
<tr>
<td>$N$</td>
<td>Rotational speed</td>
</tr>
<tr>
<td>$N_i, N_f$</td>
<td>Initial and final rotational speeds</td>
</tr>
<tr>
<td>$P$</td>
<td>Instantaneous power</td>
</tr>
<tr>
<td>$P_{avg}$</td>
<td>Average power</td>
</tr>
</tbody>
</table>

Equation Reference 4-27
Variable Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>Radius from rotation axis, or Separation distance (Law of Gravitation)</td>
</tr>
<tr>
<td>$t$</td>
<td>Time</td>
</tr>
<tr>
<td>$v$</td>
<td>Velocity</td>
</tr>
<tr>
<td>$v_f, v_{1f}, v_{2f}$</td>
<td>Final velocity</td>
</tr>
<tr>
<td>$v_i, v_{1i}$</td>
<td>Initial velocity</td>
</tr>
<tr>
<td>$W$</td>
<td>Work</td>
</tr>
<tr>
<td>$x$</td>
<td>Displacement</td>
</tr>
</tbody>
</table>

Reference: 3.

**Linear Mechanics (4, 1)**

**Equations:**

\[
F = m \cdot a \\
K_i = \frac{1}{2} m \cdot v_i^2 \\
K_f = \frac{1}{2} m \cdot v_f^2 \\
W = F \cdot x
\]

\[
W = K_f - K_i \\
P = F \cdot v \\
P_{avg} = \frac{W}{t} \\
v_f = v_i + a \cdot t
\]

**Example:**

Given: $t=10\_s$, $m=50\_lb$, $a=12.5\_ft/s^2$, $v_i=0\_ft/s$.

Solution: $v_f=125\_ft/s$, $x=625\_ft$, $F=19.4256\_lbf$, $K_i=0\_ft*lbf$, $K_f=12140.9961\_ft*lbf$, $W=12140.9961\_ft*lbf$, $P_{avg}=2.2075\_hp$. 

4-28  Equation Reference
Angular Mechanics (4, 2)

Equations:

\[ \tau = I \cdot \alpha \quad Ki = \frac{1}{2} I \cdot \omega_i^2 \quad Kf = \frac{1}{2} I \cdot \omega_f^2 \quad W = \tau \cdot \Theta \]

\[ W = Kf - Ki \quad P = \tau \cdot \omega \quad P_{avg} = \frac{W}{t} \quad \omega_f = \omega_i + \alpha \cdot t \]

\[ \alpha t = \alpha \cdot r \quad \omega = 2 \cdot \pi \cdot N \quad \omega_i = 2 \cdot \pi \cdot N_i \quad \omega_f = 2 \cdot \pi \cdot N_f \]

Example:

Given: \( I = 1750 \text{ lb} \cdot \text{in}^2 \), \( \Theta = 360^\circ \), \( r = 3.5 \text{ in} \), \( \alpha = 10.5 \text{ r/min}^2 \), \( \omega_i = 0 \text{ r/s} \).

Solution: \( \tau = 1.1017 \text{E}-3 \text{ ft} \cdot \text{lbf} \), \( Ki = 0 \text{ ft} \cdot \text{lbf} \), \( W = 6.9221 \text{E}-3 \text{ ft} \cdot \text{lbf} \), \( Kf = 6.9221 \text{E}-3 \text{ ft} \cdot \text{lbf} \), \( \alpha t = 8.5069 \text{E}-4 \text{ ft/s}^2 \), \( N_i = 0 \text{ rpm} \), \( \omega_f = 11.4868 \text{ r/min} \), \( \omega_f = 1.0940 \text{ min} \), \( N_f = 1.8282 \text{ rpm} \), \( P_{avg} = 1.9174 \text{E}-7 \text{ hp} \).

Centripetal Force (4, 3)

Equations:

\[ F = m \cdot \omega^2 \cdot r \quad \omega = \frac{v}{r} \quad ar = \frac{v^2}{r} \quad \omega = 2 \cdot \pi \cdot N \]

Example:

Given: \( m = 1 \text{ kg} \), \( r = 5 \text{ cm} \), \( N = 2000 \text{ Hz} \).

Solution: \( \omega = 12566.3706 \text{ r/s} \), \( ar = 7895683.5209 \text{ m/s} \), \( F = 7895683.5209 \text{ N} \), \( v = 628.3185 \text{ m/s} \).
Hooke's Law (4, 4)

The force is that exerted by the spring.

\[ F = -k \cdot x \quad W = \frac{-1}{2} \cdot k \cdot x^2 \]

Example:

Given: \( k = 1725 \text{ lbf/in} \), \( x = 1.25 \text{ in} \).

Solution: \( F = -2156.25 \text{ lbf} \), \( W = -112.3047 \text{ ft-lbf} \).

1D Elastic Collisions (4, 5)

Equations:

\[ v_{1f} = \frac{m_1 - m_2}{m_1 + m_2} \cdot v_{1i} \quad v_{2f} = \frac{2 \cdot m_1}{m_1 + m_2} \cdot v_{1i} \]
Example:
Given: $m_1=10\text{ kg}$, $m_2=25\text{ kg}$, $v_1i=100\text{ m/s}$.
Solution: $v_1f=-42.8571\text{ m/s}$, $v_2f=57.1429\text{ m/s}$.

Drag Force (4, 6)

Equation:

$$F = C_d \cdot \left( \frac{\rho \cdot v^2}{2} \right) \cdot A$$

Example:
Given: $C_d=0.05$, $\rho=1000\text{ kg/m}^3$, $A=7.5E6\text{ cm}^2$, $v=35\text{ m/s}$.
Solution: $F=22968750\text{ N}$.

Law of Gravitation (4, 7)

Equation:

$$F = G \cdot \left( \frac{m_1 \cdot m_2}{r^2} \right)$$

Example:
Given: $m_1=2E15\text{ kg}$, $m_2=2E18\text{ kg}$, $r=1000000\text{ km}$.
Solution: $F=266903.6\text{ N}$.

Mass-Energy Relation (4, 8)

Equation:

$$E = m \cdot c^2$$
Example:

Given: \( m=9.1\times10^{-31}\text{ kg} \).

Solution: \( E=8.1787\times10^{-14}\text{ J} \).

## Gases (5)

### Variable Names and Descriptions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )</td>
<td>Mean free path</td>
</tr>
<tr>
<td>( \rho )</td>
<td>Flow density</td>
</tr>
<tr>
<td>( \rho \theta )</td>
<td>Stagnation density</td>
</tr>
<tr>
<td>( A )</td>
<td>Flow area</td>
</tr>
<tr>
<td>( At )</td>
<td>Throat area</td>
</tr>
<tr>
<td>( d )</td>
<td>Molecular diameter</td>
</tr>
<tr>
<td>( k )</td>
<td>Specific heat ratio</td>
</tr>
<tr>
<td>( M )</td>
<td>Mach number</td>
</tr>
<tr>
<td>( m )</td>
<td>Mass</td>
</tr>
<tr>
<td>( MW )</td>
<td>Molecular weight</td>
</tr>
<tr>
<td>( n )</td>
<td>Number of moles, or Polytropic constant (Polytropic Processes)</td>
</tr>
<tr>
<td>( P )</td>
<td>Pressure, or Flow pressure (Isentropic Flow)</td>
</tr>
<tr>
<td>( P\theta )</td>
<td>Stagnation pressure</td>
</tr>
<tr>
<td>( Pc )</td>
<td>Pseudocritical pressure</td>
</tr>
<tr>
<td>( Pi, Pf )</td>
<td>Initial and final pressures</td>
</tr>
<tr>
<td>( T )</td>
<td>Temperature, or Flow temperature (Isentropic Flow)</td>
</tr>
</tbody>
</table>
Variable Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
<td>Stagnation temperature</td>
</tr>
<tr>
<td>$T_c$</td>
<td>Pseudocritical temperature</td>
</tr>
<tr>
<td>$T_i, T_f$</td>
<td>Initial and final temperatures</td>
</tr>
<tr>
<td>$V$</td>
<td>Volume</td>
</tr>
<tr>
<td>$V_i, V_f$</td>
<td>Initial and final volumes</td>
</tr>
<tr>
<td>$v_{rms}$</td>
<td>Root-mean-square (rms) velocity</td>
</tr>
<tr>
<td>$W$</td>
<td>Work</td>
</tr>
</tbody>
</table>

References: 1, 3.

Ideal Gas Law (5, 1)

Equations:

$$P \cdot V = n \cdot R \cdot T \quad m = n \cdot MW$$

Example:

Given: $T=16.85\,{}^\circ C$, $P=1\, \text{atm}$, $V=25\, \text{l}$, $MW=36\, \text{g/gmol}$.

Solution: $n=1.0506\, \text{gmol}$, $m=3.7820E^{-2}\, \text{kg}$.

Ideal Gas State Change (5, 2)

Equation:

$$\frac{P_f \cdot V_f}{T_f} = \frac{P_i \cdot V_i}{T_i}$$

Example:

Given: $P_i=1.5\, \text{kPa}$, $P_f=1.5\, \text{kPa}$, $V_i=2\, \text{l}$, $T_i=100\, {}^\circ C$, $T_f=373.15\, \text{K}$.

Solution: $V_f=2\, \text{l}$.
Isothermal Expansion (5, 3)

These equations apply to an ideal gas.

Equations:

\[ W = n \cdot R \cdot T \cdot \ln \left( \frac{V_f}{V_i} \right) \quad m = n \cdot MW \]

Example:

Given: \( V_i = 2 \text{ l} \), \( V_f = 125 \text{ l} \), \( T = 300 \degree C \), \( n = 0.25 \text{ g/mol} \), \( MW = 64 \text{ g/gmol} \).

Solution: \( W = 4926.4942 \text{ J} \), \( m = 0.016 \text{ kg} \).

Polytropic Processes (5, 4)

These equations describe a reversible pressure-volume change of an ideal gas such that \( P \cdot V^n \) is constant. Special cases include isothermal processes \( (n=1) \), isentropic processes \( (n=k, \text{ the specific heat ratio}) \), and constant-pressure processes \( (n=0) \).

Equations:

\[ \frac{P_f}{P_i} = \left( \frac{V_f}{V_i} \right)^{-n} \quad \frac{T_f}{T_i} = \left( \frac{P_f}{P_i} \right)^{\frac{n-1}{n}} \]

Example:

Given: \( P_i = 15 \text{ psi} \), \( P_f = 35 \text{ psi} \), \( V_i = 1 \text{ ft}^3 \), \( V_f = 0.50 \text{ ft}^3 \), \( T_i = 75 \degree F \).

Solution: \( n = 1.2224 \), \( T_f = 164.1117 \degree F \).
Isentropic Flow (5, 5)

The calculation differs at velocities below and above Mach 1. The Mach number is based on the speed of sound in the compressible fluid.

Equations:

\[
\begin{align*}
\frac{T}{T_0} &= \frac{2}{2 + (k - 1)M^2} \\
\frac{P}{P_0} &= \left( \frac{T}{T_0} \right)^{\frac{k}{k-1}} \\
\frac{\rho}{\rho_0} &= \left( \frac{T}{T_0} \right)^{\frac{1}{k-1}} \\
\frac{A}{At} &= \frac{1}{M} \cdot \left( \frac{2}{k+1} \cdot \left( 1 + \frac{k-1}{2}M^2 \right) \right)^{\frac{k+1}{2(k-1)}}
\end{align*}
\]

Example:

Given: \( k = 2, \ M = 0.9, \ T_0 = 26.85^\circ C, \ T = 373.15_K, \ \rho_0 = 100_{kg/m^3}, \ P_0 = 100_{kPa}, \ A = 1_{cm^2}. \)

Solution: \( P = 464.1152_{kPa}, \ At = 0.9928_{cm^2}, \ \rho = 215.4333_{kg/m^3}. \)
Real Gas Law (5, 6)

These equations adapt the ideal gas law to emulate real-gas behavior. (See “ZFACTOR” in chapter 3.)

Equations:

\[ P \cdot V = n \cdot Z \cdot R \cdot T \quad \quad m = n \cdot MW \]

Example:

Given: \( P_c = 48 \text{ atm}, T_c = 298 \text{ K}, P = 5 \text{ kPa}, V = 10 \text{ l},\) \( MW = 64 \text{ g/gmol}, T = 75 \degree \text{C}.\)

Solution: \( n = 0.0173 \text{ g/mol}, m = 1.1057 \times 10^{-3} \text{ kg}.\)

Real Gas State Change (5, 7)

This equation adapts the ideal gas state-change equation to emulate real-gas behavior. (See “ZFACTOR” in chapter 3.)

Equation:

\[ \frac{P_f \cdot V_f}{Z_f \cdot T_f} = \frac{P_i \cdot V_i}{Z_i \cdot T_i} \]

Example:

Given: \( P_c = 48 \text{ atm}, P_i = 100 \text{ kPa}, P_f = 50 \text{ kPa}, T_i = 75 \degree \text{C},\) \( T_c = 298 \text{ K}, V_i = 10 \text{ l}, T_f = 250 \degree \text{C}.\)

(Remember \( Z_f \) and \( Z_i \) are automatically calculated by the HP 48 using these variables.)

Solution: \( V_f = 30.1703 \text{ l}.\)
Kinetic Theory (5, 8)

These equations describe properties of an ideal gas.

Equations:

\[
P = \frac{n \cdot MW \cdot \text{vrms}^2}{3 \cdot V}, \quad \text{vrms} = \sqrt{\frac{3 \cdot R \cdot T}{MW}}
\]

\[
\lambda = \frac{1}{\sqrt{2 \cdot \pi} \cdot \left( \frac{n \cdot NA}{V} \right) \cdot d^2}, \quad m = n \cdot MW
\]

Example:

Given: \( P = 100 \text{ kPa}, \ V = 2 \text{ l}, \ T = 26.85 \text{ °C}, \ MW = 18 \text{ g/gmol}, \ d = 2.5 \text{ nm}. \)

Solution: \( \text{vrms} = 644.7678 \text{ m/s}, \ m = 1.4433 \times 10^{-3} \text{ kg}, \ n = 0.0802 \text{ g/mol}, \ \lambda = 1.4916 \text{ nm}. \)

Heat Transfer (6)

Variable Names and Descriptions

| \( \alpha \) | Expansion coefficient |
| \( \delta \) | Elongation |
| \( \lambda_1, \lambda_2 \) | Lower and upper wavelength limits |
| \( \lambda_{max} \) | Wavelength of maximum emissive power |
| \( \Delta T \) | Temperature difference |
| \( A \) | Area |
| \( c \) | Specific heat |
### Variable Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>( eb12 )</th>
<th>Emissive power in the range ( \lambda 1 ) to ( \lambda 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( eb )</td>
<td>Total emissive power</td>
</tr>
<tr>
<td>( f )</td>
<td>Fraction of emissive power in the range ( \lambda 1 ) to ( \lambda 2 )</td>
</tr>
<tr>
<td>( h, h1, h3 )</td>
<td>Convective heat-transfer coefficient</td>
</tr>
<tr>
<td>( k, k1, k2, k3 )</td>
<td>Thermal conductivity</td>
</tr>
<tr>
<td>( L, L1, L2, L3 )</td>
<td>Length</td>
</tr>
<tr>
<td>( m )</td>
<td>Mass</td>
</tr>
<tr>
<td>( Q )</td>
<td>Heat capacity</td>
</tr>
<tr>
<td>( q )</td>
<td>Heat transfer rate</td>
</tr>
<tr>
<td>( T )</td>
<td>Temperature</td>
</tr>
<tr>
<td>( Tc )</td>
<td>Cold surface temperature (Conduction), or Cold fluid temperature</td>
</tr>
<tr>
<td>( Th )</td>
<td>Hot surface temperature, or Hot fluid temperature (Conduction + Convection)</td>
</tr>
<tr>
<td>( Ti, Tf )</td>
<td>Initial and final temperatures</td>
</tr>
<tr>
<td>( U )</td>
<td>Overall heat transfer coefficient</td>
</tr>
</tbody>
</table>

References: 7, 9.

**Heat Capacity (6, 1)**

**Equations:**

\[
Q = m \cdot c \cdot \Delta T \\
Q = m \cdot c \cdot (T_f - T_i)
\]

**Example:**

**Given:** \( \Delta T = 15^\circ C \), \( Ti = 0^\circ C \), \( m = 10 \text{ kg} \), \( Q = 25 \text{ kJ} \).

**Solution:** \( T_f = 15^\circ C \), \( c = .1667 \text{ kJ} / (\text{kg} \cdot \text{K}) \).
Thermal Expansion (6, 2)

Equations:

\[ \delta = \alpha \cdot L \cdot \Delta T \quad \delta = \alpha \cdot L \cdot (T_f - T_i) \]

Example:

Given: \( \Delta T = 15^\circ C, L = 10\, \text{m}, T_f = 25^\circ C, \delta = 1\, \text{cm.} \)

Solution: \( T_i = 10^\circ C, \alpha = 6.6667 \times 10^{-5}/^\circ C. \)

Conduction (6, 3)

Equations:

\[ q = \frac{k \cdot A}{L} \cdot \Delta T \quad q = \frac{k \cdot A}{L} \cdot (T_h - T_c) \]
Example:

**Given:** \( T_c = 25_\degree C, \quad Th = 75_\degree C, \quad A = 12.5 \text{ m}^2, \quad L = 1.5 \text{ cm}, \quad k = .12 \text{ W/(m*K)}. \)

**Solution:** \( q = 5000 \text{ W}, \quad \Delta T = 50_\degree C. \)

**Convection (6, 4)**

![Convection Diagram]

**Equations:**

\[
q = h \cdot A \cdot \Delta T \quad \quad q = h \cdot A \cdot \left( Th - T_c \right)
\]

Example:

**Given:** \( T_c = 300 \text{ K}, \quad A = 200 \text{ m}^2, \quad h = .005 \text{ W/(m}^2\text{K)}, \quad q = 10 \text{ W}. \)

**Solution:** \( \Delta T = 10_\degree C, \quad Th = 36.8500_\degree C. \)
Conduction + Convection (6, 5)

If you have fewer than three layers, give the extra layers a zero thickness and any nonzero conductivity. The two temperatures are fluid temperatures—if instead you know a surface temperature, set the corresponding convective coefficient to $10^{4.99}$.

Equations:

$$q = \frac{A \cdot \Delta T}{\frac{1}{h_1} + \frac{L_1}{k_1} + \frac{L_2}{k_2} + \frac{L_3}{k_3} + \frac{1}{h_3}}$$

$$q = \frac{A \cdot (T_h - T_c)}{\frac{1}{h_1} + \frac{L_1}{k_1} + \frac{L_2}{k_2} + \frac{L_3}{k_3} + \frac{1}{h_3}}$$

$$U = \frac{q}{A \cdot \Delta T}$$

$$U = \frac{q}{A \cdot (T_h - T_c)}$$

Example:

Given: $\Delta T=35^\circ C$, $T_h=55^\circ C$, $A=10 \text{ m}^2$, $h_1=.05 \text{ W/(m}^2\text{K)}$, $h_3=.05 \text{ W/(m}^2\text{K)}$, $L_1=3 \text{ cm}$, $L_2=5 \text{ cm}$, $L_3=3 \text{ cm}$, $k_1=.1 \text{ W/(mK)}$, $k_2=.5 \text{ W/(mK)}$, $k_3=.1 \text{ W/(mK)}$.

Solution: $T_c=20^\circ C$, $U=0.0246 \text{ W/(m}^2\text{K)}$, $q=8.5995 \text{ W}$.
Black Body Radiation (6, 6)

See “F0λ” in chapter 3.

Equations:

\[ eb = \sigma \cdot T^4 \]
\[ f = F0\lambda \left( \lambda_2; T \right) - F0\lambda \left( \lambda_1; T \right) \]
\[ eb_{12} = f \cdot eb \]
\[ \lambda_{max} \cdot T = c3 \]
\[ q = eb \cdot A \]

Example:

Given: \( T=1000^\circ\text{C}, \lambda_1=1000\text{ nm}, \lambda_2=600\text{ nm}, A=1\text{ cm}^2 \).

Solution: \( \lambda_{max}=2276.0523\text{ nm}, eb=148984.2703\text{ W/m}^2, f=0.0036, eb_{12}=537.7264\text{ W/m}^2, q=14.8984\text{ W} \).
Magnetism (7)

Variable Names and Descriptions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu r$</td>
<td>Relative permeability</td>
</tr>
<tr>
<td>$B$</td>
<td>Magnetic field</td>
</tr>
<tr>
<td>$d$</td>
<td>Separation distance</td>
</tr>
<tr>
<td>$F_{ba}$</td>
<td>Force</td>
</tr>
<tr>
<td>$I, I_a, I_b$</td>
<td>Current</td>
</tr>
<tr>
<td>$L$</td>
<td>Length</td>
</tr>
<tr>
<td>$N$</td>
<td>Total number of turns</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of turns per unit length</td>
</tr>
<tr>
<td>$r$</td>
<td>Distance from center of wire</td>
</tr>
<tr>
<td>$r_i, r_o$</td>
<td>Inside and outside radii of toroid</td>
</tr>
<tr>
<td>$r_w$</td>
<td>Radius of wire</td>
</tr>
</tbody>
</table>

Reference: 3.

Straight Wire (7, 1)

The magnetic field calculation differs depending upon whether the point is inside or outside the wire.

Equation:

$$B = \frac{\mu_0 \cdot \mu r \cdot I}{2 \cdot \pi \cdot r}$$
Example:

Given: $\mu r=1$, $rw=.25\text{ cm}$, $r=.2\text{ cm}$, $I=25\text{ A}$.

Solution: $B=.0016\text{ T}$.

**Force between Wires (7, 2)**

The force between wires is positive for an attractive force (for currents having the same sign).

\[
F_{ba} = \frac{\mu_0 \cdot \mu_r \cdot L \cdot I_b \cdot I_a}{2 \cdot \pi \cdot d}
\]

Example:

Given: $I_a=10\text{ A}$, $I_b=20\text{ A}$, $\mu r=1$, $L=50\text{ cm}$, $d=1\text{ cm}$.

Solution: $F_{ba}=2.0000\text{E}^{-3}\text{ N}$.

**Magnetic (B) Field in Solenoid (7, 3)**
Equation:

\[ B = \mu_0 \cdot \mu r \cdot I \cdot n \]

Example:

Given: \( \mu r = 10, \ n = 50, \ I = 1.25\ A. \)

Solution: \( B = 0.0785\ T. \)

Magnetic (B) Field in Toroid (7, 4)

Equation:

\[ B = \frac{\mu_0 \cdot \mu r \cdot I \cdot N}{2 \cdot \pi} \cdot \left( \frac{2}{r_o + r_i} \right) \]

Example:

Given: \( \mu r = 10, \ N = 50, \ r_i = 5\ cm, \ r_o = 7\ cm, \ I = 10\ A. \)

Solution: \( B = 1.6667E-2\ T. \)
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Angular acceleration</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Angular velocity (Circular Motion), or Angular velocity at $t$ (Angular Motion)</td>
</tr>
<tr>
<td>$\omega_0$</td>
<td>Initial angular velocity</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Fluid density</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Angular position at $t$</td>
</tr>
<tr>
<td>$\theta_0$</td>
<td>Initial angular position (Angular Motion), or Initial vertical angle (Projectile Motion)</td>
</tr>
<tr>
<td>$a$</td>
<td>Acceleration</td>
</tr>
<tr>
<td>$A$</td>
<td>Projected horizontal area</td>
</tr>
<tr>
<td>$ar$</td>
<td>Centripetal acceleration at $r$</td>
</tr>
<tr>
<td>$Cd$</td>
<td>Drag coefficient</td>
</tr>
<tr>
<td>$m$</td>
<td>Mass</td>
</tr>
<tr>
<td>$M$</td>
<td>Planet mass</td>
</tr>
<tr>
<td>$N$</td>
<td>Rotational speed</td>
</tr>
<tr>
<td>$R$</td>
<td>Horizontal range (Projectile Motion), or Planet radius (Escape Velocity)</td>
</tr>
<tr>
<td>$r$</td>
<td>Radius</td>
</tr>
<tr>
<td>$t$</td>
<td>Time</td>
</tr>
<tr>
<td>$v$</td>
<td>Velocity at $t$ (Linear Motion), or Tangential velocity at $r$ (Circular Motion), or Terminal velocity (Terminal Velocity), or Escape velocity (Escape Velocity)</td>
</tr>
<tr>
<td>$v_0$</td>
<td>Initial velocity</td>
</tr>
<tr>
<td>$vx$</td>
<td>Horizontal component of velocity at $t$</td>
</tr>
<tr>
<td>$vy$</td>
<td>Vertical component of velocity at $t$</td>
</tr>
<tr>
<td>$x$</td>
<td>Horizontal position at $t$</td>
</tr>
<tr>
<td>$x_0$</td>
<td>Initial horizontal position</td>
</tr>
<tr>
<td>$y$</td>
<td>Vertical position at $t$</td>
</tr>
<tr>
<td>$y_0$</td>
<td>Initial vertical position</td>
</tr>
</tbody>
</table>

Reference: 3.
Linear Motion (8, 1)

Equations:
\[ x = x_0 + v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \quad x = x_0 + v \cdot t - \frac{1}{2} \cdot a \cdot t^2 \]

\[ x = x_0 + \frac{1}{2} \cdot (v_0 + v) \cdot t \quad v = v_0 + a \cdot t \]

Example:
Given: \( x_0 = 0 \text{ m}, \ x = 100 \text{ m}, \ t = 10 \text{ s}, \ v_0 = 1 \text{ m/s} \).
Solution: \( v = 19 \text{ m/s}, \ a = 1.8 \text{ m/s}^2 \).

Object in Free Fall (8, 2)

Equations:
\[ y = y_0 + v_0 \cdot t - \frac{1}{2} \cdot g \cdot t^2 \quad y = y_0 + v \cdot t + \frac{1}{2} \cdot g \cdot t^2 \]

\[ v^2 = v_0^2 - 2 \cdot g \cdot \left( y - y_0 \right) \quad v = v_0 - g \cdot t \]

Example:
Given: \( y_0 = 1000 \text{ ft}, \ y = 0 \text{ ft}, \ v_0 = 0 \text{ ft/s} \).
Solution: \( t = 7.8843 \text{ s}, \ v = -253.6991 \text{ ft/s} \).
Projectile Motion (8, 3)

Equations:

\[ x = x_0 + v_0 \cdot \cos(\theta_0) \cdot t \quad \quad y = y_0 + v_0 \cdot \sin(\theta_0) \cdot t - \frac{1}{2} g \cdot t^2 \]

\[ v_x = v_0 \cdot \cos(\theta_0) \quad \quad v_y = v_0 \cdot \sin(\theta_0) - g \cdot t \]

\[ R = \frac{v_0^2 \cdot \sin(2 \cdot \theta_0)}{g} \]

Example:

Given: \( x_0 = 0 \text{ ft}, \ y_0 = 0 \text{ ft}, \ \Theta_0 = 45^\circ, \ v_0 = 200 \text{ ft/s}, \ t = 10 \text{ s}. \)

Solution: \( R = 1243.2399 \text{ ft}, \ vx = 141.4214 \text{ ft/s}, \ vy = -180.3186 \text{ ft/s}, \ x = 1414.2136 \text{ ft}, \ y = -194.4864 \text{ ft}. \)

Angular Motion (8, 4)

Equations:

\[ \theta = \theta_0 + \omega_0 \cdot t + \frac{1}{2} \cdot \alpha \cdot t^2 \quad \quad \theta = \theta_0 + \omega \cdot t - \frac{1}{2} \cdot \alpha \cdot t^2 \]

\[ \theta = \theta_0 + \frac{1}{2} \cdot (\omega_0 + \omega) \cdot t \quad \quad \omega = \omega_0 + \alpha \cdot t \]
Example:

Given: $\theta = 0^\circ$, $\omega = 0 \text{r/min}$, $\alpha = 1.5 \text{r/min}^2$, $t = 30 \text{s}$.
Solution: $\theta = 10.7430^\circ$, $\omega = 0.7500 \text{r/min}$.

Circular Motion (8, 5)

Equations:

\[
\omega = \frac{v}{r} \quad \alpha r = \frac{v^2}{r} \quad \omega = 2 \cdot \pi \cdot N
\]

Example:

Given: $r = 25 \text{ in}$, $v = 2500 \text{ ft/s}$.
Solution: $\omega = 72000 \text{ r/min}$, $\alpha r = 3000000 \text{ ft/s}^2$, $N = 11459.1559 \text{ rpm}$.

Terminal Velocity (8, 6)

Equation:

\[
v = \sqrt{\frac{2 \cdot m \cdot g}{C_d \cdot \rho \cdot A}}
\]

Example:

Given: $C_d = 0.15$, $\rho = 0.025 \text{ lb/ft}^3$, $A = 100000 \text{ in}^2$, $m = 1250 \text{ lb}$.
Solution: $v = 1757.4709 \text{ ft/s}$.

Escape Velocity (8, 7)

Equation:

\[
v = \sqrt{\frac{2 \cdot G \cdot M}{R}}
\]
Example:

Given: \( M=1.5\times10^23\text{ lb}, R=5000\text{ mi} \).

Solution: \( v=3485.1106\text{ ft/s} \).

Optics (9)

<table>
<thead>
<tr>
<th>Variable Names and Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \theta_1 )</td>
</tr>
<tr>
<td>( \theta_2 )</td>
</tr>
<tr>
<td>( \theta_B )</td>
</tr>
<tr>
<td>( \theta_c )</td>
</tr>
<tr>
<td>( f )</td>
</tr>
<tr>
<td>( m )</td>
</tr>
<tr>
<td>( n,n_1,n_2 )</td>
</tr>
<tr>
<td>( r,r_1,r_2 )</td>
</tr>
<tr>
<td>( u )</td>
</tr>
<tr>
<td>( v )</td>
</tr>
</tbody>
</table>

For reflection and refraction problems, the focal length and radius of curvature are positive in the direction of the outgoing light (reflected or refracted). The object distance is positive in front of the surface. The image distance is positive in the direction of the outgoing light (reflected or refracted). The magnification is positive for an upright image.

Reference: 3.
Law of Refraction (9, 1)

Equation:

\[ n_1 \cdot \sin(\theta_1) = n_2 \cdot \sin(\theta_2) \]

Example:

Given: \( n_1 = 1 \), \( n_2 = 1.333 \), \( \theta_1 = 45^\circ \).

Solution: \( \theta_2 = 32.0367^\circ \).

Critical Angle (9, 2)

Equation:

\[ \sin(\theta_c) = \frac{n_1}{n_2} \]
Example:

Given: $n_1 = 1, n_2 = 1.5$.

Solution: $\theta_c = 41.8103^\circ$.

Brewster's Law (9, 3)

The Brewster angle is the angle of incidence at which the reflected wave is completely polarized.

Equations:

$$\tan(\theta_B) = \frac{n_2}{n_1} \quad \theta_B + \theta_2 = 90$$

Example:

Given: $n_1 = 1, n_2 = 1.5$.

Solution: $\theta_B = 56.3099^\circ, \theta_2 = 33.6901^\circ$.

Spherical Reflection (9, 4)
Equations:

\[ \frac{1}{u} + \frac{1}{v} = \frac{1}{f} \quad f = \frac{1}{2} \cdot r \quad m = \frac{-v}{u} \]

Example:

Given: \( u = 10 \text{ cm}, \ v = 300 \text{ cm}, \ r = 19.35 \text{ cm}. \)

Solution: \( m = -30, \ f = 9.6774 \text{ cm}. \)

**Spherical Refraction (9, 5)**

![Spherical Refraction Diagram]

Equation:

\[ \frac{n_1}{u} + \frac{n_2}{v} = \frac{n_2 - n_1}{r} \]

Example:

Given: \( u = 8 \text{ cm}, \ v = 12 \text{ cm}, \ r = 2 \text{ cm}, \ n_I = 1. \)

Solution: \( n_2 = 1.5000. \)
Thin Lens (9, 6)

$r_1$ is for the front surface, and $r_2$ is for the back surface.

**Equations:**

\[
\frac{1}{u} + \frac{1}{v} = \frac{1}{f} \\
\frac{1}{f} = (n - 1) \cdot \left( \frac{1}{r_1} - \frac{1}{r_2} \right) \\
m = \frac{-v}{u}
\]

**Example:**

Given: $r_1=5 \text{ cm}$, $r_2=20 \text{ cm}$, $n=1.5$, $u=50 \text{ cm}$.

Solution: $f=13.3333 \text{ cm}$, $v=18.1818 \text{ cm}$, $m=-.3636$.

**Oscillations (10)**

**Variable Names and Descriptions**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega$</td>
<td>Angular frequency</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Phase angle</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Cone angle</td>
</tr>
<tr>
<td>$a$</td>
<td>Acceleration at $t$</td>
</tr>
<tr>
<td>$f$</td>
<td>Frequency</td>
</tr>
<tr>
<td>$G$</td>
<td>Shear modulus of elasticity</td>
</tr>
<tr>
<td>$h$</td>
<td>Cone height</td>
</tr>
</tbody>
</table>
Variable Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I$</td>
<td>Moment of inertia</td>
</tr>
<tr>
<td>$J$</td>
<td>Polar moment of inertia</td>
</tr>
<tr>
<td>$k$</td>
<td>Spring constant</td>
</tr>
<tr>
<td>$L$</td>
<td>Length of pendulum</td>
</tr>
<tr>
<td>$m$</td>
<td>Mass</td>
</tr>
<tr>
<td>$t$</td>
<td>Time</td>
</tr>
<tr>
<td>$T$</td>
<td>Period</td>
</tr>
<tr>
<td>$v$</td>
<td>Velocity at $t$</td>
</tr>
<tr>
<td>$x$</td>
<td>Displacement at $t$</td>
</tr>
<tr>
<td>$x_m$</td>
<td>Displacement amplitude</td>
</tr>
</tbody>
</table>

Reference: 3.

Mass-Spring System (10, 1)

Equations:

$$\omega = \sqrt{\frac{k}{m}} \quad T = \frac{2\pi}{\omega} \quad \omega = 2\pi f$$

Example:

Given: $k=20\text{ N/m}$, $m=5\text{ kg}$.

Solution: $\omega=2\text{ r/s}$, $T=3.1416\text{ s}$, $f=.3183\text{ Hz}$.
Simple Pendulum (10, 2)

Equations:

$$\omega = \sqrt{\frac{g}{L}} \quad T = \frac{2 \cdot \pi}{\omega} \quad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: \(L = 15\) _cm_.

Solution: \(\omega = 8.0856\_r/s, \quad T = 0.7771\_s, \quad f = 1.2869\_Hz\).

Conical Pendulum (10, 3)

Equations:

$$\omega = \sqrt{\frac{g}{h}} \quad h = L \cdot \cos \left( \theta \right) \quad T = \frac{2 \cdot \pi}{\omega}$$

$$\omega = 2 \cdot \pi \cdot f$$
Example:

Given: \( L=25\text{ cm}, \ h=20\text{ cm} \).

Solution: \( \theta=36.899^\circ, \ T=.8973\text{ s}, \ \omega=7.0024\_r/s, \ f=1.1145\text{ Hz} \).

**Torsional Pendulum (10, 4)**

\[
\omega = \sqrt{\frac{G \cdot J}{L \cdot I}} \quad \text{Torsion Angle: } T = \frac{2 \cdot \pi}{\omega} \quad \omega = 2 \cdot \pi \cdot f
\]

Example:

Given: \( G=1000\text{ kPa}, \ J=17\text{ mm}^4, \ L=26\text{ cm}, \ I=50\text{ kg} \cdot \text{m}^2 \).

Solution: \( \omega=1.1435E\_3\_r/s, \ f=1.8200E\_4\_\text{Hz}, \ T=5494.4862\_\text{s} \).

**Simple Harmonic (10, 5)**

Equations:

\[
x = x_m \cdot \cos \left( \omega \cdot t + \phi \right) \quad v = -\omega \cdot x_m \cdot \sin \left( \omega \cdot t + \phi \right) \\
a = -\omega^2 \cdot x_m \cdot \cos \left( \omega \cdot t + \phi \right) \quad \omega = 2 \cdot \pi \cdot f
\]
Example:

Given: \( zm = 10 \text{ cm}, \omega = 15 \text{ r/s}, \phi = 25^\circ, t = 25 \mu s. \)

Solution: \( x = 9.0615 \text{ cm}, v = -0.6344 \text{ m/s}, a = -20.3884 \text{ m/s}^2, \)
\( f = 2.3873 \text{ Hz}. \)

---

**Plane Geometry (11)**

**Variable Names and Descriptions**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \beta )</td>
<td>Central angle of polygon</td>
</tr>
<tr>
<td>( \theta )</td>
<td>Vertex angle of polygon</td>
</tr>
<tr>
<td>( A )</td>
<td>Area</td>
</tr>
<tr>
<td>( b )</td>
<td>Base length (Rectangle, Triangle), or Length of semiaxis in ( x ) direction (Ellipse)</td>
</tr>
<tr>
<td>( C )</td>
<td>Circumference</td>
</tr>
<tr>
<td>( d )</td>
<td>Distance to rotation axis in ( y ) direction</td>
</tr>
<tr>
<td>( h )</td>
<td>Height (Rectangle, Triangle), or Length of semiaxis in ( y ) direction (Ellipse)</td>
</tr>
<tr>
<td>( I_x, I_y )</td>
<td>Moment of inertia about ( x ) axis</td>
</tr>
<tr>
<td>( I_d )</td>
<td>Moment of inertia in ( x ) direction at ( d )</td>
</tr>
<tr>
<td>( I_y )</td>
<td>Moment of inertia about ( y ) axis</td>
</tr>
<tr>
<td>( J )</td>
<td>Polar moment of inertia at centroid</td>
</tr>
<tr>
<td>( L )</td>
<td>Side length of polygon</td>
</tr>
<tr>
<td>( n )</td>
<td>Number of sides</td>
</tr>
<tr>
<td>( P )</td>
<td>Perimeter</td>
</tr>
<tr>
<td>( r )</td>
<td>Radius</td>
</tr>
<tr>
<td>( r_i, r_o )</td>
<td>Inside and outside radii</td>
</tr>
<tr>
<td>( r_s )</td>
<td>Distance to side of polygon</td>
</tr>
<tr>
<td>( r_v )</td>
<td>Distance to vertex of polygon</td>
</tr>
<tr>
<td>( v )</td>
<td>Horizontal distance to vertex</td>
</tr>
</tbody>
</table>

Reference: 4.
Circle (11, 1)

Equations:

\[ A = \pi r^2 \quad C = 2\pi r \quad I = \frac{\pi r^4}{4} \]

\[ J = \frac{\pi r^4}{2} \quad I_d = I + A \cdot d^2 \]

Example:

Given: \( r = 5 \text{ cm}, \ d = 1.5 \text{ cm} \).

Solution: \( C = 31.4159 \text{ cm}, \ A = 78.5398 \text{ cm}^2, \ I = 4908738.5 \text{ mm}^4, \ J = 9817477.0 \text{ mm}^4, \ I_d = 6675884.4 \text{ mm}^4 \).

Ellipse (11, 2)
Equations:

\[ A = \pi \cdot b \cdot h \quad C = 2 \cdot \pi \cdot \sqrt{\frac{b^2 + h^2}{2}} \quad I = \frac{\pi \cdot b \cdot h^3}{4} \]

\[ J = \frac{\pi \cdot b \cdot h}{4} \cdot \left( b^2 + h^2 \right) \quad Id = I + A \cdot d^2 \]

Example:

Given: \( b = 17.85 \mu\text{m}, h = 78.9725 \mu\text{in}, d = 0.00000012 \text{ft} \).

Solution: \( A = 1.1249 \times 10^{-6} \text{cm}^2, C = 7.9805 \times 10^{-3} \text{cm}, I = 1.1315 \times 10^{-10} \text{mm}^4, J = 9.0733 \times 10^{-9} \text{mm}^4, Id = 1.1330 \times 10^{-10} \text{mm}^4 \).

Rectangle (11, 3)

Equations:

\[ A = b \cdot h \quad P = 2b + 2h \quad I = \frac{b \cdot h^3}{12} \]

\[ J = \frac{b \cdot h}{12} \cdot \left( b^2 + h^2 \right) \quad Id = I + A \cdot d^2 \]

Example:

Given: \( b = 4 \text{_chain}, h = 7 \text{_rd}, d = 39.26 \text{_in} \).

Set guesses for \( I, J, \) and \( Id \) in \( \text{km}^4 \).

Solution: \( A = 28328108.2691 \text{_cm}^2, P = 23134.3662 \text{_cm}, I = 2.9257 \times 10^{-7} \text{_km}^4, J = 1.8211 \times 10^{-6} \text{_km}^4, Id = 2.9539 \times 10^{-7} \text{_km}^4 \).
Regular Polygon (11, 4)

Equations:

\[
A = \frac{1}{4} \cdot n \cdot L^2 \quad \text{TAN} \left( \frac{180}{n} \right) \\
P = n \cdot L \\
r_s = \frac{L}{2} \quad \text{TAN} \left( \frac{180}{n} \right) \\
r_v = \frac{L}{2} \quad \text{SIN} \left( \frac{180}{n} \right) \\
\theta = \frac{n - 2}{n} \cdot 180 \\
\beta = \frac{360}{n}
\]

Example:

Given: \( n=8, \ L=.5 \text{ yd} \).

Solution: \( A=10092.9501 \text{ cm}^2, \ P=365.7600 \text{ cm}, \ r_s=55.1889 \text{ cm}, \ r_v=59.7361 \text{ cm}, \ \theta=135^\circ, \ \beta=45^\circ \).

Circular Ring (11, 5)
Equations:

\[ A = \pi \cdot \left( r_o^2 - r_i^2 \right) \quad I = \frac{\pi}{4} \cdot \left( r_o^4 - r_i^4 \right) \]

\[ J = \frac{\pi}{2} \cdot \left( r_o^4 - r_i^4 \right) \quad \text{Id} = I + A \cdot d^2 \]

Example:

Given: \( r_o = 4 \, \mu \text{m}, \ r_i = 25.0 \, \text{kÅ}, \ d = 0.1 \, \text{mil} \).

Solution: \( A = 3.0631 \times 10^{-7} \, \text{cm}^2, \ I = 1.7038 \times 10^{-10} \, \text{mm}^4, \ J = 3.4076 \times 10^{-10} \, \text{mm}^4, \ \text{Id} = 3.0648 \times 10^{-10} \, \text{mm}^4 \).

Triangle (11, 6)

![Diagram of a triangle with labels and equations](Image)

Equations:

\[ A = \frac{b \cdot h}{2} \quad P = b + \sqrt{v^2 + h^2 + \sqrt{(b - v)^2 + h^2}} \]

\[ I_x = \frac{b \cdot h^3}{36} \quad I_y = \frac{b \cdot h}{36} \cdot \left( b^2 - b \cdot v + v^2 \right) \]

\[ J = \frac{b \cdot h}{36} \cdot \left( h^2 + b^2 - b \cdot v + v^2 \right) \quad \text{Id} = I_x + A \cdot d^2 \]
Example:

Given: $h=4.33012781892\text{ in}$, $v=2.5\text{ in}$, $P=15\text{ in}$, $d=2\text{ in}$.

Solution: $b=5.0000\text{ in}$, $I_x=11.2764\text{ in}^4$, $I_y=11.2764\text{ in}^4$, $J=22.5527\text{ in}^4$, $A=10.8253\text{ in}^2$, $Id=54.5776\text{ in}^4$. 

Solid Geometry (12)

Variable Names and Descriptions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>Total surface area</td>
</tr>
<tr>
<td>$b$</td>
<td>Base length</td>
</tr>
<tr>
<td>$d$</td>
<td>Distance to rotation axis in $z$ direction</td>
</tr>
<tr>
<td>$h$</td>
<td>Height in $z$ direction (Cone, Cylinder), or Height in $y$ direction (Parallelepiped)</td>
</tr>
<tr>
<td>$I_x, I_{xx}$</td>
<td>Moment of inertia about $x$ axis</td>
</tr>
<tr>
<td>$I_d$</td>
<td>Moment of inertia in $x$ direction at $d$</td>
</tr>
<tr>
<td>$I_{zz}$</td>
<td>Moment of inertia about $z$ axis</td>
</tr>
<tr>
<td>$m$</td>
<td>Mass</td>
</tr>
<tr>
<td>$r$</td>
<td>Radius</td>
</tr>
<tr>
<td>$t$</td>
<td>Thickness in $z$ direction</td>
</tr>
<tr>
<td>$V$</td>
<td>Volume</td>
</tr>
</tbody>
</table>

Reference: 4.
Cone (12, 1)

Equations:

\[ V = \frac{\pi}{3} r^2 h \]
\[ A = \pi r^2 + \pi r \sqrt{r^2 + h^2} \]

\[ I_{xx} = \frac{3}{20} m r^2 + \frac{3}{80} m h^2 \]
\[ I_{zz} = \frac{3}{10} m r^2 \]

\[ I_d = I_{xx} + m d^2 \]

Example:

Given: \( r = 7 \text{ cm}, h = 12.5 \text{ cm}, m = 12.25 \text{ kg}, d = 3.5 \text{ cm} \).

Solution: \( V = 641.4085 \text{ cm}^3 \), \( A = 468.9953 \text{ cm}^2 \), \( I_{xx} = 0.0162 \text{ kg} \cdot \text{m}^2 \), \( I_{zz} = 0.0180 \text{ kg} \cdot \text{m}^2 \), \( I_d = 0.0312 \text{ kg} \cdot \text{m}^2 \).

Cylinder (12, 2)
Equations:

\[ V = \pi \cdot r^2 \cdot h \quad A = 2 \cdot \pi \cdot r^2 + 2 \cdot \pi \cdot r \cdot h \]

\[ I_{xx} = \frac{1}{4} \cdot m \cdot r^2 + \frac{1}{12} \cdot m \cdot h^2 \quad I_{zz} = \frac{1}{2} \cdot m \cdot r^2 \]

\[ I_d = I_{xx} + m \cdot d^2 \]

Example:

Given: \( r=8.5\, \text{in}, \, h=65\, \text{in}, \, m=12000\, \text{lbs}, \, d=2.5\, \text{in}. \)

Solution: \( V=14753.7045\, \text{in}^3, \, A=3925.4200\, \text{in}^2, \)

\( I_{xx}=4441750\, \text{lb}\cdot\text{in}^2, \, I_{zz}=433500\, \text{lb}\cdot\text{in}^2, \, I_d=4516750\, \text{lb}\cdot\text{in}^2. \)

Parallelepiped (12, 3)

Equations:

\[ V = b \cdot h \cdot t \]
\[ A = 2 \cdot \left( b \cdot h + b \cdot t + h \cdot t \right) \]
\[ I = \frac{1}{12} \cdot m \cdot \left( h^2 + t^2 \right) \quad I_d = I + m \cdot d^2 \]
Example:

Given: $b=36 \text{ in}$, $h=12 \text{ in}$, $t=72 \text{ in}$, $m=83 \text{ lb}$, $d=7 \text{ in}$.

Solution: $V=31104 \text{ in}^3$, $A=7776 \text{ in}^2$, $I=36852 \text{ lb}\cdot\text{in}^2$, $Id=40919 \text{ lb}\cdot\text{in}^2$.

Sphere (12, 4)

Equations:

\[ V = \frac{4}{3} \cdot \pi \cdot r^3 \quad A = 4 \cdot \pi \cdot r^2 \quad I = \frac{2}{5} \cdot m \cdot r^2 \quad Id = I + m \cdot d^2 \]

Example:

Given: $d=14 \text{ cm}$, $m=3.75 \text{ kg}$, $Id=486.5 \text{ lb}\cdot\text{in}^2$.

Solution: $r=21.4273 \text{ cm}$, $V=41208.7268 \text{ cm}^3$, $A=5769.5719 \text{ cm}^2$, $I=0.0689 \text{ kg}\cdot\text{m}^2$. 
## Variable Names and Descriptions

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_F$</td>
<td>Forward common-base current gain</td>
</tr>
<tr>
<td>$\alpha_R$</td>
<td>Reverse common-base current gain</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Body factor</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Modulation parameter</td>
</tr>
<tr>
<td>$\mu_n$</td>
<td>Electron mobility</td>
</tr>
<tr>
<td>$\phi_p$</td>
<td>Fermi potential</td>
</tr>
<tr>
<td>$\Delta L$</td>
<td>Length adjustment (PN Step Junctions), or Channel encroachment (NMOS Transistors)</td>
</tr>
<tr>
<td>$\Delta W$</td>
<td>Width adjustment (PN Step Junctions), or Width contraction (NMOS Transistors)</td>
</tr>
<tr>
<td>$a$</td>
<td>Channel thickness</td>
</tr>
<tr>
<td>$A_j$</td>
<td>Effective junction area</td>
</tr>
<tr>
<td>$B V$</td>
<td>Breakdown voltage</td>
</tr>
<tr>
<td>$C_j$</td>
<td>Junction capacitance per unit area</td>
</tr>
<tr>
<td>$C_{ox}$</td>
<td>Silicon dioxide capacitance per unit area</td>
</tr>
<tr>
<td>$E_{1}$</td>
<td>Breakdown-voltage field factor</td>
</tr>
<tr>
<td>$E_{max}$</td>
<td>Maximum electric field</td>
</tr>
<tr>
<td>$G_0$</td>
<td>Channel conductance</td>
</tr>
<tr>
<td>$g_{ds}$</td>
<td>Output conductance</td>
</tr>
<tr>
<td>$g_m$</td>
<td>Transconductance</td>
</tr>
<tr>
<td>$I$</td>
<td>Diode current</td>
</tr>
<tr>
<td>$I_B$</td>
<td>Total base current</td>
</tr>
<tr>
<td>$I_C$</td>
<td>Total collector current</td>
</tr>
<tr>
<td>$I_{CEO}$</td>
<td>Collector current (collector-to-base open)</td>
</tr>
<tr>
<td>$I_{CO}$</td>
<td>Collector current (emitter-to-base open)</td>
</tr>
<tr>
<td>$I_{CS}$</td>
<td>Collector-to-base saturation current</td>
</tr>
<tr>
<td>$I_{D,IDS}$</td>
<td>Drain current</td>
</tr>
<tr>
<td>$I_E$</td>
<td>Total emitter current</td>
</tr>
<tr>
<td>$I_{ES}$</td>
<td>Emitter-to-base saturation current</td>
</tr>
</tbody>
</table>
### Variable Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Transistor saturation current</td>
</tr>
<tr>
<td>J</td>
<td>Current density</td>
</tr>
<tr>
<td>Js</td>
<td>Saturation current density</td>
</tr>
<tr>
<td>L</td>
<td>Drawn mask length (PN Step Junctions), or Drawn gate length (NMOS Transistors), or Channel length (JFETs)</td>
</tr>
<tr>
<td>Le</td>
<td>Effective gate length</td>
</tr>
<tr>
<td>NA</td>
<td>P-side doping (PN Step Junctions), or Substrate doping (NMOS Transistors)</td>
</tr>
<tr>
<td>ND</td>
<td>N-side doping (PN Step Junctions), or N-channel doping (JFETs)</td>
</tr>
<tr>
<td>T</td>
<td>Temperature</td>
</tr>
<tr>
<td>tox</td>
<td>Gate silicon dioxide thickness</td>
</tr>
<tr>
<td>Va</td>
<td>Applied voltage</td>
</tr>
<tr>
<td>VBC</td>
<td>Base-to-collector voltage</td>
</tr>
<tr>
<td>VBE</td>
<td>Base-to-emitter voltage</td>
</tr>
<tr>
<td>Vbi</td>
<td>Built-in voltage</td>
</tr>
<tr>
<td>VBS</td>
<td>Substrate voltage</td>
</tr>
<tr>
<td>VCEsat</td>
<td>Collector-to-emitter saturation voltage</td>
</tr>
<tr>
<td>VDS</td>
<td>Applied drain voltage</td>
</tr>
<tr>
<td>VDsat</td>
<td>Saturation voltage</td>
</tr>
<tr>
<td>VGS</td>
<td>Applied gate voltage</td>
</tr>
<tr>
<td>Vt</td>
<td>Threshold voltage</td>
</tr>
<tr>
<td>Vt0</td>
<td>Threshold voltage (at zero substrate voltage)</td>
</tr>
<tr>
<td>W</td>
<td>Drawn mask width (PN Step Junctions), or Drawn width (NMOS Transistors), or Channel width (JFETs)</td>
</tr>
<tr>
<td>We</td>
<td>Effective width</td>
</tr>
<tr>
<td>xd</td>
<td>Depletion-region width</td>
</tr>
<tr>
<td>xdm</td>
<td>Depletion-layer width</td>
</tr>
<tr>
<td>xj</td>
<td>Junction depth</td>
</tr>
</tbody>
</table>

References: 5, 8.
PN Step Junctions (13, 1)

These equations for a silicon PN-junction diode use a “two-sided step-junction” model—the doping density changes abruptly at the junction. The equations assume the current density is determined by minority carriers injected across the depletion region and the PN junction is rectangular in its layout. The temperature should be between 77 and 500 K. (See “SIDENS” in chapter 3.)
Equations:

\[ V_{bi} = \frac{k \cdot T}{q} \cdot \ln \left( \frac{NA \cdot ND}{ni^2} \right) \]

\[ xd = \sqrt{\frac{2 \cdot \epsilon_s \cdot \epsilon_0}{q} \cdot (V_{bi} - V_a) \cdot \left( \frac{1}{NA} + \frac{1}{ND} \right)} \]

\[ C_j = \frac{\epsilon_s \cdot \epsilon_0}{xd} \]

\[ E_{max} = \frac{2 \cdot (V_{bi} - V_a)}{xd} \]

\[ BV = \frac{\epsilon_s \cdot \epsilon_0 \cdot E_l^2}{2 \cdot q} \cdot \left( \frac{1}{NA} + \frac{1}{ND} \right) \]

\[ J = J_s \cdot \left( \frac{q \cdot V_a}{k \cdot T} - 1 \right) \]

\[ A_j = \left( W + 2 \cdot \Delta W \right) \cdot \left( L + 2 \cdot \Delta L \right) + \pi \cdot \left( W + L + 2 \cdot \Delta W + 2 \cdot \Delta L \right) \cdot x_j + 2 \cdot \pi \cdot x_j^2 \]

\[ I = J \cdot A_j \]

Example:

Given: \( ND=1E22 \text{ cm}^{-3}, \ NA=1E15 \text{ 1/cm}^3, \ T=26.85^\circ \text{C}, \)
\( J_s=1E-6 \text{ pA/cm}^2, \ V_a=-20 \text{ V}, \ E_l=33E5 \text{ V/cm}, \ W=10 \mu, \)
\( \Delta W=1 \mu, \ L=10 \mu, \ \Delta L=1 \mu, \ x_j=2 \mu. \)

Solution: \( V_{bi}=0.9962 \text{ V}, \ xd=5.2551 \mu, \ C_j=2005.0141 \text{ pF/cm}^2, \)
\( E_{max}=79908.5240 \text{ V/cm}, \ BV=358.0825 \text{ V}, \ J=-1.0E-12 \text{ A/cm}^2, \)
\( A_j=3.1993E-6 \text{ cm}^2, \ I=-3.1993E-15 \text{ mA}. \)
NMOS Transistors (13, 2)

These equations for a silicon NMOS transistor use a two-port network model. They include linear and nonlinear regions in the device characteristics and are based on a gradual-channel approximation (the electric fields in the direction of current flow are small compared to those perpendicular to the flow). The drain current and transconductance calculations differ depending on whether the transistor is in the linear, saturated, or cutoff region. The equations assume the physical geometry of the device is a rectangle, second-order length-parameter effects are negligible, short-channel, hot-carrier, and velocity-saturation effects are negligible, and subthreshold currents are negligible. (See “SIDENS Function” in chapter 3.)
Equations:

\[ W_e = W - 2 \cdot \Delta W \]
\[ L_e = L - 2 \cdot \Delta L \]
\[ C_{ox} = \frac{\epsilon_{ox} \cdot \epsilon_0}{t_{ox}} \]

\[ I_{DS} = C_{ox} \cdot \mu_n \cdot \left( \frac{W_e}{L_e} \right) \cdot \left( (V_{GS} - V_t) \cdot V_{DS} - \frac{V_{DS}^2}{2} \right) \cdot (1 + \lambda \cdot V_{DS}) \]

\[ \gamma = \sqrt{2 \cdot \epsilon_{si} \cdot \epsilon_0 \cdot q \cdot N_A} \]

\[ V_t = V_{t0} + \gamma \cdot \left( \sqrt{2 \cdot \text{ABS}(\phi_p) + \text{ABS}(V_{BS}) - \sqrt{2 \cdot \text{ABS}(\phi_p)}} \right) \]

\[ \phi_p = \frac{-k \cdot T}{q} \cdot \ln \left( \frac{N_A}{n_i} \right) \]

\[ g_{ds} = I_{DS} \cdot \lambda \]

\[ g_m = \sqrt{C_{ox} \cdot \mu_n \cdot \left( \frac{W_e}{L_e} \right) \cdot \left( 1 + \lambda \cdot V_{DS} \right) \cdot 2 \cdot I_{DS}} \]

\[ V_{DSat} = V_{GS} - V_t \]

Example:

Given: \( t_{ox}=700\ \text{Å}, \quad N_A=1\times10^{15} \text{cm}^{-3}, \quad \mu_n=600 \text{cm}^2/(V\cdot s), \quad T=26.85^\circ C, \quad V_{t0}=0.75 \text{V}, \quad V_{GS}=5 \text{V}, \quad V_{BS}=0 \text{V}, \quad V_{DS}=5 \text{V}, \quad W=25 \mu m, \quad \Delta W=1 \mu m, \quad L=4 \text{m}, \quad \Delta L=75 \mu m, \quad \lambda=0.05 \text{V}^{-1}. \)

Solution: \( W_e=23 \mu m, \quad L_e=2.5 \mu m, \quad C_{ox}=49330.4750 \text{pF/cm}^2, \quad \gamma=3725 \text{V}^{-1.5}, \quad \phi_p=-0.2898 \text{V}, \quad V_{t0}=0.75 \text{V}, \quad V_{DSat}=4.25 \text{V}, \quad I_{DS}=3.0741 \text{mA}, \quad g_{ds}=1.5370E-4 \text{S}, \quad g_m=1.4466 \text{mA/V}. \)
Bipolar Transistors (13, 3)

These equations for an NPN silicon bipolar transistor are based on large-signal models developed by J.J. Ebers and J.L. Moll. The offset-voltage calculation differs depending on whether the transistor is saturated or not. The equations also include the special conditions when the emitter-base or collector-base junction is open, which are convenient for measuring transistor parameters.

Equations:

\[
IE = -IES \cdot \left( \frac{q \cdot VBE}{e^{\frac{k \cdot T}{T}} - 1} \right) + \alpha R \cdot ICS \cdot \left( \frac{q \cdot VBC}{e^{\frac{k \cdot T}{T}} - 1} \right)
\]

\[
IC = -ICS \cdot \left( \frac{q \cdot VBC}{e^{\frac{k \cdot T}{T}} - 1} \right) + \alpha F \cdot IES \cdot \left( \frac{q \cdot VBE}{e^{\frac{k \cdot T}{T}} - 1} \right)
\]

\[
IS = \alpha F \cdot IES \quad IS = \alpha R \cdot ICS \quad IB + IE + IC = 0
\]

\[
ICO = ICS \cdot \left( 1 - \alpha F \cdot \alpha R \right) \quad ICEO = \frac{ICO}{1 - \alpha F}
\]

\[
VCE_{\text{sat}} = \frac{k \cdot T}{q \cdot \ln} \left[ \frac{1 + \frac{IC}{IB} \cdot \left( 1 - \alpha R \right)}{\alpha R \cdot \left( 1 - \alpha R \right) \cdot \left( 1 - \frac{IC}{IB} \cdot \frac{1 - \alpha F}{\alpha F} \right)} \right]
\]

Equation Reference 4-73
Example:

Given: \( I_{ES} = 1 \times 10^{-5} \text{nA}, \ I_{CS} = 2 \times 10^{-5} \text{nA}, \ T = 26.85 ^\circ\text{C}, \ \alpha F = 0.98, \ \alpha R = 0.49, \ I_C = 1 \text{mA}, \ V_{BC} = -10 \text{V}. \)

Solution: \( V_{BE} = 0.6553 \text{V}, \ I_S = 0.0000098 \text{nA}, \ I_{CO} = 0.00010396 \text{nA}, \ I_{CEO} = 0.0005198 \text{nA}, \ I_E = -1.0204 \text{mA}, \ I_B = 0.0204 \text{mA}, \ V_{CEsat} = 0 \text{V}. \)

**JFETs (13, 4)**

These equations for a silicon N-channel junction field-effect transistor (JFET) are based on the single-sided step-junction approximation, which assumes the gates are heavily doped compared to the channel doping. The drain-current calculation differs depending on whether the gate-junction depletion-layer thickness is less than or greater than the channel thickness. The equations assume the channel is uniformly doped and end effects (such as contact, drain, and source resistances) are negligible. (See “SIDENS” in chapter 3.)
Equations:

\[ V_{bi} = \frac{k \cdot T}{q} \cdot \ln \left( \frac{ND}{ni} \right) \]

\[ x_{\text{dmax}} = \sqrt{\frac{2 \cdot \varepsilon \cdot \varepsilon_0}{q \cdot ND}} \cdot \left( V_{bi} - V_{GS} + V_{DS} \right) \]

\[ G_0 = q \cdot ND \cdot \mu n \cdot \left( \frac{a \cdot W}{L} \right) \]

\[ I_D = G_0 \cdot \left( V_{DS} - \frac{2}{3} \sqrt{\frac{2 \cdot \varepsilon \cdot \varepsilon_0}{q \cdot ND \cdot a^2}} \cdot \left( \left( V_{bi} - V_{GS} + V_{DS} \right)^{\frac{3}{2}} - \left( V_{bi} - V_{GS} \right)^{\frac{3}{2}} \right) \right) \]

\[ V_{DSat} = \frac{q \cdot ND \cdot a^2}{2 \cdot \varepsilon \cdot \varepsilon_0} - \left( V_{bi} - V_{GS} \right) \]

\[ V_t = V_{bi} - \frac{q \cdot ND \cdot a^2}{2 \cdot \varepsilon \cdot \varepsilon_0} \]

\[ g_m = G_0 \cdot \left( 1 - \sqrt{\frac{2 \cdot \varepsilon \cdot \varepsilon_0}{q \cdot ND \cdot a^2}} \cdot \left( V_{bi} - V_{GS} \right) \right) \]

Example:

Given: \( ND=1E16 \text{ } \text{ } \text{ } \text{ } _{1/cm^3}, \ W=6 \mu , \ a=1 \mu , \ L=2 \mu , \)
\( \mu n=1248 \text{ } \text{ } \text{ } \text{ } _{cm^2/(Vs)}, \ VGS=-4 \text{ } \text{ } V, \ VDS=4 \text{ } \text{ } V, \ T=26.85 \text{ } ^\circ \text{C}. \)

Solution: \( V_{bi}=.3493 \text{ } \text{ } V, \ x_{\text{dmax}}=1.0479 \mu , \ G_0=5.9986E-4 \text{ } \text{ } S, \)
\( I_D=.2268 \text{ } \text{ } mA, \ V_{DSat}=3.2537 \text{ } \text{ } V, \ V_t=-7.2537 \text{ } \text{ } V, \ g_m=.1462 \text{ } \text{ } mA/V. \)
## Stress Analysis (14)

### Variable Names and Descriptions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td>Elongation</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>Normal strain</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Shear strain</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Angle of twist</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Normal stress</td>
</tr>
<tr>
<td>$\sigma_1$</td>
<td>Maximum principal normal stress</td>
</tr>
<tr>
<td>$\sigma_2$</td>
<td>Minimum principal normal stress</td>
</tr>
<tr>
<td>$\sigma_{avg}$</td>
<td>Normal stress on plane of maximum shear stress</td>
</tr>
<tr>
<td>$\sigma_x$</td>
<td>Normal stress in x direction</td>
</tr>
<tr>
<td>$\sigma_{x1}$</td>
<td>Normal stress in rotated-x direction</td>
</tr>
<tr>
<td>$\sigma_y$</td>
<td>Normal stress in y direction</td>
</tr>
<tr>
<td>$\sigma_{y1}$</td>
<td>Normal stress in rotated-y direction</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Shear stress</td>
</tr>
<tr>
<td>$\tau_{max}$</td>
<td>Maximum shear stress</td>
</tr>
<tr>
<td>$\tau_{x1y1}$</td>
<td>Rotated shear stress</td>
</tr>
<tr>
<td>$\tau_{xy}$</td>
<td>Shear stress</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Rotation angle</td>
</tr>
<tr>
<td>$\theta_{p_1}$</td>
<td>Angle to plane of maximum principal normal stress</td>
</tr>
<tr>
<td>$\theta_{p_2}$</td>
<td>Angle to plane of minimum principal normal stress</td>
</tr>
<tr>
<td>$\theta_s$</td>
<td>Angle to plane of maximum shear stress</td>
</tr>
<tr>
<td>$A$</td>
<td>Area</td>
</tr>
<tr>
<td>$E$</td>
<td>Modulus of elasticity</td>
</tr>
<tr>
<td>$G$</td>
<td>Shear modulus of elasticity</td>
</tr>
<tr>
<td>$J$</td>
<td>Polar moment of inertia</td>
</tr>
<tr>
<td>$L$</td>
<td>Length</td>
</tr>
<tr>
<td>$P$</td>
<td>Load</td>
</tr>
<tr>
<td>$r$</td>
<td>Radius</td>
</tr>
<tr>
<td>$T$</td>
<td>Torque</td>
</tr>
</tbody>
</table>

Reference: 2.

4-76  Equation Reference
Normal Stress (14, 1)

Equations:

\[ \sigma = E \cdot \varepsilon \quad \varepsilon = \frac{\delta}{L} \quad \sigma = \frac{P}{A} \]

Example:

Given: \( P=40000 \text{ lbf}, L=1 \text{ ft}, A=3.14159265359 \text{ in}^2, E=10E6 \text{ psi} \).

Solution: \( \delta=0.0153 \text{ in}, \varepsilon=0.0013, \sigma=12732.3954 \text{ psi} \).

Shear Stress (14, 2)

Equations:

\[ \tau = G \cdot \gamma \quad \gamma = \frac{r \cdot \phi}{L} \quad \tau = \frac{T \cdot r}{J} \]
Example:

Given: $L=6\text{ ft}$, $r=2\text{ in}$, $J=10.4003897419\text{ in}^4$, $G=12000000\text{ psi}$, $\tau=12000\text{ psi}$.

Solution: $T=5200.1949\text{ ft}\cdot\text{lbf}$, $\phi=2.0626\degree$, $\gamma=5.7296\times10^{-2}\degree$.

Stress on an Element (14, 3)

Stresses and strains are positive in the directions shown.

Equations:

\[
\sigma_{x1} = \frac{\sigma_x + \sigma_y}{2} + \frac{\sigma_x - \sigma_y}{2} \cdot \cos(2\theta) + \tau_{xy} \cdot \sin(2\theta)
\]

\[
\sigma_{x1} + \sigma_{y1} = \sigma_x + \sigma_y
\]

\[
\tau_{x1y1} = -\left(\frac{\sigma_x - \sigma_y}{2}\right) \cdot \sin(2\theta) + \tau_{xy} \cdot \cos(2\theta)
\]

Example:

Given: $\sigma_x=15000\text{ kPa}$, $\sigma_y=4755\text{ kPa}$, $\tau_{xy}=7500\text{ kPa}$, $\theta=30\degree$.

Solution: $\sigma_{x1}=18933.9405\text{ kPa}$, $\sigma_{y1}=821.0595\text{ kPa}$, $\tau_{x1y1}=-686.2151\text{ kPa}$.
Mohr’s Circle (14, 4)

Equations:

\[ \sigma_1 = \frac{\sigma_x + \sigma_y}{2} + \sqrt{\left( \frac{\sigma_x - \sigma_y}{2} \right)^2 + \tau_{xy}^2} \]

\[ \sigma_1 + \sigma_2 = \sigma_x + \sigma_y \]

\[ \sin(2 \cdot \theta_1) = \frac{\tau_{xy}}{\sqrt{\left( \frac{\sigma_x - \sigma_y}{2} \right)^2 + \tau_{xy}^2}} \]

\[ \theta_2 = \theta_1 + 90 \quad \tau_{\text{max}} = \frac{\sigma_1 - \sigma_2}{2} \]

\[ \theta_s = \theta_1 - 45 \quad \sigma_{\text{avg}} = \frac{\sigma_x + \sigma_y}{2} \]

Example:

Given: \( \sigma_x = -5600 \, \text{psi}, \sigma_y = -18400 \, \text{psi}, \tau_{xy} = 4800 \, \text{psi}. \)

Solution: \( \sigma_1 = -4000 \, \text{psi}, \sigma_2 = -20000 \, \text{psi}, \theta_1 = 18.4349 ^\circ, \theta_2 = 108.4349 ^\circ, \tau_{\text{max}} = 8000 \, \text{psi}, \theta_s = -26.5651 ^\circ, \sigma_{\text{avg}} = -12000 \, \text{psi}. \)
Waves (15)

Variable Names and Descriptions

| β   | Sound level          |
| λ   | Wavelength           |
| ω   | Angular frequency    |
| ρ   | Density of medium    |
| B   | Bulk modulus of elasticity |
| f   | Frequency            |
| I   | Sound intensity      |
| k   | Angular wave number  |
| s   | Longitudinal displacement at x and t |
| sm  | Longitudinal amplitude |
| t   | Time                 |
| v   | Speed of sound in medium (Sound Waves), or Wave speed (Transverse Waves, Longitudinal Waves) |
| x   | Position             |
| y   | Transverse displacement at x and t |
| ym  | Transverse amplitude |

Reference: 3.

Transverse Waves (15, 1)

Equations:

\[ y = y_m \cdot \sin \left( k \cdot x - \omega \cdot t \right) \quad v = \lambda \cdot f \quad k = \frac{2 \cdot \pi}{\lambda} \quad \omega = 2 \cdot \pi \cdot f \]
Example:

Given: \( y_m = 6.37\, \text{cm}, k = 32.11\, \text{r/cm}, x = 0.03\, \text{cm}, \omega = 7000\, \text{r/s}, t = 1\, \text{s} \).

Solution: \( f = 1114.0846\, \text{Hz}, \lambda = 0.0020\, \text{cm}, y_\perp = 2.6655\, \text{cm}, \)
\( v = 218.0006\, \text{cm/s} \).

**Longitudinal Waves (15, 2)**

Equations:

\[
\begin{align*}
    s &= x \cdot \cos \left( k \cdot x - \omega \cdot t \right) \\
    v &= \omega \cdot f \\
    k &= \frac{2 \cdot \pi}{\lambda} \\
    \omega &= 2 \cdot \pi \cdot f
\end{align*}
\]

Example:

Given: \( sm = 6.37\, \text{cm}, k = 32.11\, \text{r/cm}, x = 0.03\, \text{cm}, \omega = 7000\, \text{r/s}, t = 1\, \text{s} \).

Solution: \( s = 5.7855\, \text{cm}, v = 2.1800\, \text{m/s}, \lambda = 0.1957\, \text{cm}, \\
\lambda = 1114.0846\, \text{Hz} \).

**Sound Waves (15, 3)**

Equations:

\[
\begin{align*}
    v &= \sqrt{\frac{B}{\rho}} \\
    I &= \frac{1}{2} \cdot \rho \cdot v \cdot \omega^2 \cdot \text{sm}^2 \\
    \beta &= 10 \cdot \text{LOG} \left( \frac{I}{10} \right) \\
    \omega &= 2 \cdot \pi \cdot f
\end{align*}
\]

Example:

Given: \( sm = 10\, \text{cm}, \omega = 6000\, \text{r/s}, B = 12500\, \text{kPa}, \rho = 65\, \text{kg/m}^3 \).

Solution: \( v = 438.5290\, \text{m/s}, I = 5130789412.97\, \text{W/m}^2, \\
\beta = 217.1018\, \text{dB}, f = 954.9297\, \text{Hz} \).
References


Error and Status Messages

In the following tables, messages are first arranged alphabetically by name and then numerically by message number.

---

### Messages Listed Alphabetically

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
<th># (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledged</td>
<td>Alarm acknowledged.</td>
<td>619</td>
</tr>
<tr>
<td>All Variables Known</td>
<td>No unknowns to solve for.</td>
<td>E405</td>
</tr>
<tr>
<td>Autoscaling</td>
<td>Calculator is autoscaling x- and/or y-axis.</td>
<td>610</td>
</tr>
<tr>
<td>Awaiting Server Cmd.</td>
<td>Indicates Server mode active.</td>
<td>C0C</td>
</tr>
<tr>
<td>Bad Argument Type</td>
<td>One or more stack arguments were incorrect type for operation.</td>
<td>202</td>
</tr>
<tr>
<td>Bad Argument Value</td>
<td>Argument value out of operation's range.</td>
<td>203</td>
</tr>
<tr>
<td>Bad Guess(es)</td>
<td>Guess(es) supplied to HP Solve application or ROOT lie outside domain of equation.</td>
<td>A01</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Bad Packet Block check</td>
<td>Computed packet checksum doesn’t match checksum in packet.</td>
<td>C01</td>
</tr>
<tr>
<td>Can't Edit Null Char.</td>
<td>Attempted to edit a string containing character ( \equiv ) (character code 0).</td>
<td>102</td>
</tr>
<tr>
<td>Circular Reference</td>
<td>Attempted to store a variable name into itself.</td>
<td>129</td>
</tr>
<tr>
<td>Connecting</td>
<td>Indicates verifying IR or serial connection.</td>
<td>C0A</td>
</tr>
<tr>
<td>Constant?</td>
<td>HP Solve application or ROOT returned same value at every sample point of current equation.</td>
<td>A02</td>
</tr>
<tr>
<td>Copied to stack</td>
<td>( \Rightarrow \text{STK} ) copied selected equation to stack.</td>
<td>623</td>
</tr>
<tr>
<td>Current equation:</td>
<td>Identifies current equation.</td>
<td>608</td>
</tr>
<tr>
<td>Deleting Column</td>
<td>MatrixWriter application is deleting a column.</td>
<td>504</td>
</tr>
<tr>
<td>Deleting Row</td>
<td>MatrixWriter application is deleting a row.</td>
<td>503</td>
</tr>
<tr>
<td>Directory Not Allowed</td>
<td>Name of existing directory variable used as argument.</td>
<td>12A</td>
</tr>
<tr>
<td>Directory Recursion</td>
<td>Attempted to store a directory into itself.</td>
<td>002</td>
</tr>
<tr>
<td>Empty catalog</td>
<td>No data in current catalog (Equation, Statistics, Alarm).</td>
<td>60D</td>
</tr>
<tr>
<td>Empty stack</td>
<td>The stack contains no data.</td>
<td>C15</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Enter alarm, press SET</td>
<td>Alarm entry prompt.</td>
<td>61A</td>
</tr>
<tr>
<td>Enter eqn, press NEW</td>
<td>Store new equation in ( EQ ).</td>
<td>60A</td>
</tr>
<tr>
<td>Enter value (zoom out if &gt;1), press ENTER</td>
<td>Zoom operations prompt.</td>
<td>622</td>
</tr>
<tr>
<td>EQ Invalid for MINIT</td>
<td>( EQ ) must contain at least two equations (or programs) and two variables.</td>
<td>E403</td>
</tr>
<tr>
<td>Extremum</td>
<td>Result returned by HP Solve application or ROOT is an extremum rather than a root.</td>
<td>A06</td>
</tr>
<tr>
<td>HALT Not Allowed</td>
<td>A program containing HALT executed while MatrixWriter application, DRAW, or HP Solve application active.</td>
<td>126</td>
</tr>
<tr>
<td>I/O setup menu</td>
<td>Identifies I/O setup menu.</td>
<td>61C</td>
</tr>
<tr>
<td>Illegal During MROOT</td>
<td>Multiple-Equation Solver command attempted during MROOT execution.</td>
<td>E406</td>
</tr>
<tr>
<td>Implicit () off</td>
<td>Implicit parentheses off.</td>
<td>207</td>
</tr>
<tr>
<td>Implicit () on</td>
<td>Implicit parentheses on.</td>
<td>208</td>
</tr>
<tr>
<td>Incomplete Subexpression</td>
<td>( \uparrow ), ( \downarrow ), or ( \text{ENTER} ) pressed before all function arguments supplied.</td>
<td>206</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Inconsistent Units</td>
<td>Attempted unit conversion with incompatible units.</td>
<td>B02</td>
</tr>
<tr>
<td>Infinite Result</td>
<td>Math exception: Calculation such as 1/0 infinite result.</td>
<td>305</td>
</tr>
<tr>
<td>Inserting Column</td>
<td>MatrixWriter application is inserting a column.</td>
<td>506</td>
</tr>
<tr>
<td>Inserting Row</td>
<td>MatrixWriter application is inserting a row.</td>
<td>505</td>
</tr>
<tr>
<td>Insufficient Memory</td>
<td>Not enough free memory to execute operation.</td>
<td>001</td>
</tr>
<tr>
<td>Insufficient Σ Data</td>
<td>A Statistics command was executed when ΣDAT did not contain enough data points for calculation.</td>
<td>603</td>
</tr>
<tr>
<td>Interrupted</td>
<td>The HP Solve application or ROOT was interrupted by [CANCEL].</td>
<td>A03</td>
</tr>
<tr>
<td>Invalid Array Element</td>
<td>[ENTER] returned object of wrong type for current matrix.</td>
<td>502</td>
</tr>
<tr>
<td>Invalid Card Data</td>
<td>HP 48 does not recognize data on plug-in card.</td>
<td>008</td>
</tr>
<tr>
<td>Invalid Date</td>
<td>Date argument not real number in correct format, or was out of range.</td>
<td>D01</td>
</tr>
<tr>
<td>Invalid Definition</td>
<td>Incorrect structure of equation argument for DEFINE.</td>
<td>12C</td>
</tr>
<tr>
<td>Invalid Dimension</td>
<td>Array argument had wrong dimensions.</td>
<td>501</td>
</tr>
</tbody>
</table>

A-4  Error and Status Messages
Messages Listed Alphabetically (continued)

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
<th># (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid EQ</td>
<td>Attempted operation from PICTURE FCN menu when ( EQ ) did not contain algebraic, or, attempted DRAW with CONIC plot type when ( EQ ) did not contain algebraic.</td>
<td>607</td>
</tr>
<tr>
<td>Invalid IOPAR</td>
<td>( IOPAR ) not a list, or one or more objects in list missing or invalid.</td>
<td>C12</td>
</tr>
<tr>
<td>Invalid Mpar</td>
<td>( Mpar ) variable not created by MINIT.</td>
<td>E401</td>
</tr>
<tr>
<td>Invalid Name</td>
<td>Received illegal filename, or server asked to send illegal filename.</td>
<td>C17</td>
</tr>
<tr>
<td>Invalid PPAR</td>
<td>( PPAR ) not a list, or one or more objects in list missing or invalid.</td>
<td>12E</td>
</tr>
<tr>
<td>Invalid PRTPAR</td>
<td>( PRTPAR ) not a list, or one or more objects in list missing or invalid.</td>
<td>C13</td>
</tr>
<tr>
<td>Invalid PTYPE</td>
<td>Plot type invalid for current equation.</td>
<td>620</td>
</tr>
<tr>
<td>Invalid Repeat</td>
<td>Alarm repeat interval out of range.</td>
<td>D03</td>
</tr>
<tr>
<td>Invalid Server</td>
<td>Invalid command received while in Server mode.</td>
<td>C08</td>
</tr>
<tr>
<td>Cmd.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalid Syntax</td>
<td>HP 48 unable execute ( \text{ENTER} ), OBJ(\rightarrow ), or STR(\rightarrow ) due to invalid object syntax.</td>
<td>106</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Invalid Time</td>
<td>Time argument not real number in correct format, or out of range.</td>
<td>D02</td>
</tr>
<tr>
<td>Invalid Unit</td>
<td>Unit operation attempted with invalid or undefined user unit.</td>
<td>B01</td>
</tr>
<tr>
<td>Invalid User Function</td>
<td>Type or structure of object executed as user-defined function was incorrect.</td>
<td>103</td>
</tr>
<tr>
<td>Invalid Σ Data</td>
<td>Statistics command executed with invalid object stored in ΣDAT.</td>
<td>601</td>
</tr>
<tr>
<td>Invalid Σ Data LH(Neg)</td>
<td>Non-linear curve fit attempted when ΣDAT matrix contained a negative element.</td>
<td>605</td>
</tr>
<tr>
<td>Invalid Σ Data LN(0)</td>
<td>Non-linear curve fit attempted when ΣDAT matrix contained a 0 element.</td>
<td>606</td>
</tr>
<tr>
<td>Invalid ΣPAR</td>
<td>ΣPAR not list, or one or more objects in list missing or invalid.</td>
<td>604</td>
</tr>
<tr>
<td>Keyword Conflict</td>
<td>A plug-in card conflicts with an equation library variable. Remove the card to continue.</td>
<td>E303</td>
</tr>
<tr>
<td>LAST CMD Disabled</td>
<td>(CMD) pressed while that recovery feature disabled.</td>
<td>125</td>
</tr>
<tr>
<td>LAST STACK Disabled</td>
<td>(UNDO) pressed while that recovery feature disabled.</td>
<td>124</td>
</tr>
<tr>
<td>LASTARG Disabled</td>
<td>(ARG) executed while that recovery feature disabled.</td>
<td>205</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Low Battery</td>
<td>System batteries too low to safely print or perform I/O.</td>
<td>C14</td>
</tr>
<tr>
<td>Memory Clear</td>
<td>HP 48 memory was cleared.</td>
<td>005</td>
</tr>
<tr>
<td>Name Conflict</td>
<td>Execution of</td>
<td>(where) attempted to assign value to variable of integration or summation index.</td>
</tr>
<tr>
<td>Name the equation, press ENTER</td>
<td>Name equation and store it in $EQ$.</td>
<td>60B</td>
</tr>
<tr>
<td>Name the statistics data, press ENTER</td>
<td>Name statistics data and store it in $\Sigma{DAT}$.</td>
<td>621</td>
</tr>
<tr>
<td>Negative Underflow</td>
<td>Math exception: Calculation returned negative, non-zero result greater than $-\text{MINR}$.</td>
<td>302</td>
</tr>
<tr>
<td>No Current Equation</td>
<td>$\text{SOLVR}$, $\text{DRAW}$, or $\text{RCEQ}$ executed with nonexistent $EQ$.</td>
<td>104</td>
</tr>
<tr>
<td>No current equation.</td>
<td>Plot and HP Solve application status message.</td>
<td>609</td>
</tr>
<tr>
<td>No Picture Available</td>
<td>No picture is included for the selected equation.</td>
<td>E304</td>
</tr>
<tr>
<td>No Room in Port</td>
<td>Insufficient free memory in specified RAM port.</td>
<td>00B</td>
</tr>
<tr>
<td>No Room to Save Stack</td>
<td>Not enough free memory to save copy of the stack. LAST STACK is automatically disabled.</td>
<td>101</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>No Room to Show Stack</td>
<td>Stack objects displayed by type only due to low memory condition.</td>
<td>131</td>
</tr>
<tr>
<td>No stat data to plot</td>
<td>No data stored in $\Sigma$DAT.</td>
<td>60F</td>
</tr>
<tr>
<td>Non-Empty Directory</td>
<td>Attempted to purge non-empty directory.</td>
<td>12B</td>
</tr>
<tr>
<td>Non-Real Result</td>
<td>Execution of HP Solve application, ROOT, DRAW, or $\int$ returned result other than real number or unit.</td>
<td>12F</td>
</tr>
<tr>
<td>Nonexistent Alarm</td>
<td>Alarm list did not contain alarm specified by alarm command.</td>
<td>D04</td>
</tr>
<tr>
<td>Nonexistent $\Sigma$DAT</td>
<td>Statistics command executed when $\Sigma$DAT did not exist.</td>
<td>602</td>
</tr>
<tr>
<td>Object Discarded</td>
<td>Sender sent an EOF (Z) packet with a “D” in the data field.</td>
<td>C0F</td>
</tr>
<tr>
<td>Object In Use</td>
<td>Attempted PURGE or STO into a backup object when its stored object was in use.</td>
<td>009</td>
</tr>
<tr>
<td>Object Not in Port</td>
<td>Attempted to access a nonexistent backup object or library.</td>
<td>00C</td>
</tr>
<tr>
<td>(OFF SCREEN)</td>
<td>Function value, root, extremum, or intersection was not visible in current display.</td>
<td>61F</td>
</tr>
</tbody>
</table>
## Messages Listed Alphabetically (continued)

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
<th># (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out of Memory</td>
<td>One or more objects must be purged to continue calculator operation.</td>
<td>135</td>
</tr>
<tr>
<td>Overflow</td>
<td>Math exception: Calculation returned result greater in absolute value than MAXR.</td>
<td>303</td>
</tr>
<tr>
<td>Packet #</td>
<td>Indicates packet number during send or receive.</td>
<td>C10</td>
</tr>
<tr>
<td>Parity Error</td>
<td>Received bytes' parity bit doesn't match current parity setting.</td>
<td>C05</td>
</tr>
<tr>
<td>Plot Type:</td>
<td>Label introducing current plot type.</td>
<td>61D</td>
</tr>
<tr>
<td>Port Closed</td>
<td>Possible I/R or serial hardware failure. Run self-test.</td>
<td>C09</td>
</tr>
<tr>
<td>Port Not Available</td>
<td>Used a port command on an empty port, or one containing ROM instead of RAM.</td>
<td>00A</td>
</tr>
<tr>
<td></td>
<td>Attempted to execute a server command that itself uses the I/O port.</td>
<td></td>
</tr>
<tr>
<td>Positive Underflow</td>
<td>Math exception: Calculation returned positive, non-zero result less than MINR.</td>
<td>301</td>
</tr>
<tr>
<td>Power Lost</td>
<td>Calculator turned on following a power loss. Memory may have been corrupted.</td>
<td>006</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Processing Command</td>
<td>Indicates processing of host command packet.</td>
<td>C11</td>
</tr>
<tr>
<td>Protocol Error</td>
<td>Received a packet whose length was shorter than a null packet.</td>
<td>C07</td>
</tr>
<tr>
<td></td>
<td>Maximum packet length parameter from other machine is illegal.</td>
<td></td>
</tr>
<tr>
<td>Receive Buffer Overrun</td>
<td>Kermit: More than 255 bytes of retries sent before HP 48 received another packet.</td>
<td>C04</td>
</tr>
<tr>
<td></td>
<td>SRECV: Incoming data overflowed the buffer.</td>
<td></td>
</tr>
<tr>
<td>Receive Error</td>
<td>UART overrun or framing error.</td>
<td>C03</td>
</tr>
<tr>
<td>Receiving</td>
<td>Identifies object name while receiving.</td>
<td>C0E</td>
</tr>
<tr>
<td>Retry #</td>
<td>Indicates number of retries while retrying packet exchange.</td>
<td>C0B</td>
</tr>
<tr>
<td>Select a model</td>
<td>Select statistics curve fitting model.</td>
<td>614</td>
</tr>
<tr>
<td>Select plot type</td>
<td>Select plot type.</td>
<td>60C</td>
</tr>
<tr>
<td>Select repeat interval</td>
<td>Select alarm repeat interval.</td>
<td>61B</td>
</tr>
<tr>
<td>Sending</td>
<td>Identifies object name while sending.</td>
<td>C0D</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Sign Reversal</td>
<td>HP Solve application or ROOT unable to find point at which current equation evaluates to zero, but did find two neighboring points at which equation changed sign.</td>
<td>A05</td>
</tr>
<tr>
<td>Single Equation</td>
<td>Only one equation supplied to Multiple-Equation Solver. Use HP Solve application.</td>
<td>E402</td>
</tr>
<tr>
<td>Timeout</td>
<td>Printing to serial port: Received XOFF and timed out waiting for XON. Kermit: Timed out waiting for packet to arrive.</td>
<td>C02</td>
</tr>
<tr>
<td>Too Few Arguments</td>
<td>Command required more arguments than were available on stack.</td>
<td>201</td>
</tr>
<tr>
<td>Too Many Unknowns</td>
<td>Multiple Equation Solver can’t calculate a value given the current knowns. Supply another value or add an equation.</td>
<td>E404</td>
</tr>
<tr>
<td>Transfer Failed</td>
<td>Ten successive attempts to receive a good packet were unsuccessful.</td>
<td>C06</td>
</tr>
<tr>
<td>Unable to find root</td>
<td>PROOT is unable to determine all roots of the polynomial.</td>
<td>C001</td>
</tr>
<tr>
<td>Unable to Isolate</td>
<td>ISOL failed because specified name absent or contained in argument a function with no inverse.</td>
<td>130</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
<td># (hex)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Undefined Local Name</td>
<td>Executed or recalled local name for which corresponding local variable did not exist.</td>
<td>003</td>
</tr>
<tr>
<td>Undefined Name</td>
<td>Executed or recalled global name for which corresponding variable does not exist.</td>
<td>204</td>
</tr>
<tr>
<td>Undefined Result</td>
<td>Calculation such as 0/0 generated mathematically undefined result.</td>
<td>304</td>
</tr>
<tr>
<td>Undefined XLIB Name</td>
<td>Executed an XLIB name when specified library absent.</td>
<td>004</td>
</tr>
<tr>
<td>Warning:</td>
<td>Label introducing current status message.</td>
<td>007</td>
</tr>
<tr>
<td>Wrong Argument Count</td>
<td>User-defined function evaluated with an incorrect number of parenthetical arguments.</td>
<td>128</td>
</tr>
<tr>
<td>x and y-axis zoom.</td>
<td>Identifies zoom option.</td>
<td>627</td>
</tr>
<tr>
<td>x axis zoom.</td>
<td>Identifies zoom option.</td>
<td>625</td>
</tr>
<tr>
<td>x axis zoom w/AUTO.</td>
<td>Identifies zoom option.</td>
<td>624</td>
</tr>
<tr>
<td>y axis zoom.</td>
<td>Identifies zoom option.</td>
<td>626</td>
</tr>
<tr>
<td>ZERO</td>
<td>Result returned by the HP Solve application or ROOT is a root (a point at which current equation evaluates to zero).</td>
<td>A04</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>Identifies no execution action when EXECES pressed.</td>
<td>61E</td>
</tr>
</tbody>
</table>
## Messages Listed Numerically

<table>
<thead>
<tr>
<th># (hex)</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Insufficient Memory</td>
</tr>
<tr>
<td>002</td>
<td>Directory Recursion</td>
</tr>
<tr>
<td>003</td>
<td>Undefined Local Name</td>
</tr>
<tr>
<td>004</td>
<td>Undefined XLIB Name</td>
</tr>
<tr>
<td>005</td>
<td>Memory Clear</td>
</tr>
<tr>
<td>006</td>
<td>Power Lost</td>
</tr>
<tr>
<td>008</td>
<td>Invalid Card Data</td>
</tr>
<tr>
<td>009</td>
<td>Object In use</td>
</tr>
<tr>
<td>00A</td>
<td>Port Not Available</td>
</tr>
<tr>
<td>00B</td>
<td>No Room in Port</td>
</tr>
<tr>
<td>00C</td>
<td>Object Not in Port</td>
</tr>
<tr>
<td>101</td>
<td>No Room to Save Stack</td>
</tr>
<tr>
<td>102</td>
<td>Can't Edit Null Char.</td>
</tr>
<tr>
<td>103</td>
<td>Invalid User Function</td>
</tr>
<tr>
<td>104</td>
<td>No Current Equation</td>
</tr>
<tr>
<td>106</td>
<td>Invalid Syntax</td>
</tr>
<tr>
<td>124</td>
<td>LAST STACK Disabled</td>
</tr>
<tr>
<td>125</td>
<td>LAST CMD Disabled</td>
</tr>
<tr>
<td>126</td>
<td>HALT Not Allowed</td>
</tr>
<tr>
<td>128</td>
<td>Wrong Argument Count</td>
</tr>
<tr>
<td>129</td>
<td>Circular Reference</td>
</tr>
<tr>
<td>12A</td>
<td>Directory Not Allowed</td>
</tr>
<tr>
<td>12B</td>
<td>Non-Empty Directory</td>
</tr>
<tr>
<td>12C</td>
<td>Invalid Definition</td>
</tr>
<tr>
<td>12E</td>
<td>Invalid PPAR</td>
</tr>
<tr>
<td>12F</td>
<td>Non-Real Result</td>
</tr>
<tr>
<td>130</td>
<td>Unable to Isolate</td>
</tr>
<tr>
<td>131</td>
<td>No Room to Show Stack</td>
</tr>
</tbody>
</table>

### Out-of-Memory Prompts

<table>
<thead>
<tr>
<th># (hex)</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>135</td>
<td>Out of Memory</td>
</tr>
<tr>
<td>13C</td>
<td>Name Conflict</td>
</tr>
<tr>
<td># (hex)</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>Stack Errors</strong></td>
</tr>
<tr>
<td>201</td>
<td>Too Few Arguments</td>
</tr>
<tr>
<td>202</td>
<td>Bad Argument Type</td>
</tr>
<tr>
<td>203</td>
<td>Bad Argument Value</td>
</tr>
<tr>
<td>204</td>
<td>Undefined Name</td>
</tr>
<tr>
<td>205</td>
<td>LASTARG Disabled</td>
</tr>
<tr>
<td></td>
<td><strong>EquationWriter Application Messages</strong></td>
</tr>
<tr>
<td>206</td>
<td>Incomplete Subexpression</td>
</tr>
<tr>
<td>207</td>
<td>Implicit () off</td>
</tr>
<tr>
<td>208</td>
<td>Implicit () on</td>
</tr>
<tr>
<td></td>
<td><strong>Floating-Point Errors</strong></td>
</tr>
<tr>
<td>301</td>
<td>Positive Underflow</td>
</tr>
<tr>
<td>302</td>
<td>Negative Underflow</td>
</tr>
<tr>
<td>303</td>
<td>Overflow</td>
</tr>
<tr>
<td>304</td>
<td>Undefined Result</td>
</tr>
<tr>
<td>305</td>
<td>Infinite Result</td>
</tr>
<tr>
<td></td>
<td><strong>Array Messages</strong></td>
</tr>
<tr>
<td>501</td>
<td>Invalid Dimension</td>
</tr>
<tr>
<td>502</td>
<td>Invalid Array Element</td>
</tr>
<tr>
<td>503</td>
<td>Deleting Row</td>
</tr>
<tr>
<td>504</td>
<td>Deleting Column</td>
</tr>
<tr>
<td>505</td>
<td>Inserting Row</td>
</tr>
<tr>
<td>506</td>
<td>Inserting Column</td>
</tr>
<tr>
<td></td>
<td><strong>Statistics Messages</strong></td>
</tr>
<tr>
<td>601</td>
<td>Invalid Σ Data</td>
</tr>
<tr>
<td>602</td>
<td>Nonexistent ΣDAT</td>
</tr>
<tr>
<td>603</td>
<td>Insufficient Σ Data</td>
</tr>
<tr>
<td>604</td>
<td>Invalid ΣPAR</td>
</tr>
<tr>
<td>605</td>
<td>Invalid Σ Data LH(Neg)</td>
</tr>
<tr>
<td>606</td>
<td>Invalid Σ Data LH(0)</td>
</tr>
</tbody>
</table>
### Messages Listed Numerically (continued)

<table>
<thead>
<tr>
<th># (hex)</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>607</td>
<td>Invalid EQ</td>
</tr>
<tr>
<td>608</td>
<td>Current equation:</td>
</tr>
<tr>
<td>609</td>
<td>No current equation.</td>
</tr>
<tr>
<td>60A</td>
<td>Enter eqn, press NEW</td>
</tr>
<tr>
<td>60B</td>
<td>Name the equation, press ENTER</td>
</tr>
<tr>
<td>60C</td>
<td>Select plot type</td>
</tr>
<tr>
<td>60D</td>
<td>Empty catalog</td>
</tr>
<tr>
<td>60F</td>
<td>No stat data to plot</td>
</tr>
<tr>
<td>610</td>
<td>Autoscaling</td>
</tr>
<tr>
<td>614</td>
<td>Select a model</td>
</tr>
<tr>
<td>619</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>61A</td>
<td>Enter alarm, press SET</td>
</tr>
<tr>
<td>61B</td>
<td>Select repeat interval</td>
</tr>
<tr>
<td>61C</td>
<td>I/O setup menu</td>
</tr>
<tr>
<td>61D</td>
<td>Plot type:</td>
</tr>
<tr>
<td>61E</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>61F</td>
<td>(OFF SCREEN)</td>
</tr>
<tr>
<td>620</td>
<td>Invalid PTYPE</td>
</tr>
<tr>
<td>621</td>
<td>Name the stat data, press ENTER</td>
</tr>
<tr>
<td>622</td>
<td>Enter value (zoom out if &gt;1), press ENTER</td>
</tr>
<tr>
<td>623</td>
<td>Copied to stack</td>
</tr>
<tr>
<td>624</td>
<td>x axis zoom w/AUTO.</td>
</tr>
<tr>
<td>625</td>
<td>x axis zoom.</td>
</tr>
<tr>
<td>626</td>
<td>y axis zoom.</td>
</tr>
<tr>
<td>627</td>
<td>x and y-axis zoom.</td>
</tr>
<tr>
<td>A01</td>
<td>Bad Guess(es)</td>
</tr>
<tr>
<td>A02</td>
<td>Constant?</td>
</tr>
<tr>
<td>A03</td>
<td>Interrupted</td>
</tr>
<tr>
<td>A04</td>
<td>Zero</td>
</tr>
<tr>
<td>A05</td>
<td>Sign Reversal</td>
</tr>
<tr>
<td>A06</td>
<td>Extremum</td>
</tr>
</tbody>
</table>
### Messages Listed Numerically (continued)

<table>
<thead>
<tr>
<th># (hex)</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit Management</strong></td>
<td></td>
</tr>
<tr>
<td>B01</td>
<td>Invalid Unit</td>
</tr>
<tr>
<td>B02</td>
<td>Inconsistent Units</td>
</tr>
<tr>
<td><strong>I/O and Printing</strong></td>
<td></td>
</tr>
<tr>
<td>C01</td>
<td>Bad Packet Block check</td>
</tr>
<tr>
<td>C02</td>
<td>Timeout</td>
</tr>
<tr>
<td>C03</td>
<td>Receive Error</td>
</tr>
<tr>
<td>C04</td>
<td>Receive Buffer Overrun</td>
</tr>
<tr>
<td>C05</td>
<td>Parity Error</td>
</tr>
<tr>
<td>C06</td>
<td>Transfer Failed</td>
</tr>
<tr>
<td>C07</td>
<td>Protocol Error</td>
</tr>
<tr>
<td>C08</td>
<td>Invalid Server Cmd.</td>
</tr>
<tr>
<td>C09</td>
<td>Port Closed</td>
</tr>
<tr>
<td>C0A</td>
<td>Connecting</td>
</tr>
<tr>
<td>C0B</td>
<td>Retry #</td>
</tr>
<tr>
<td>C0C</td>
<td>Awaiting Server Cmd.</td>
</tr>
<tr>
<td>C0D</td>
<td>Sending</td>
</tr>
<tr>
<td>C0E</td>
<td>Receiving</td>
</tr>
<tr>
<td>C0F</td>
<td>Object Discarded</td>
</tr>
<tr>
<td>C10</td>
<td>Packet #</td>
</tr>
<tr>
<td>C11</td>
<td>Processing Command</td>
</tr>
<tr>
<td>C12</td>
<td>Invalid IOPAR</td>
</tr>
<tr>
<td>C13</td>
<td>Invalid PRTPAR</td>
</tr>
<tr>
<td>C14</td>
<td>Low Battery</td>
</tr>
<tr>
<td>C15</td>
<td>Empty Stack</td>
</tr>
<tr>
<td>C17</td>
<td>Invalid Name</td>
</tr>
<tr>
<td># (hex)</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>Time Messages</strong></td>
</tr>
<tr>
<td>D01</td>
<td>Invalid Date</td>
</tr>
<tr>
<td>D02</td>
<td>Invalid Time</td>
</tr>
<tr>
<td>D03</td>
<td>Invalid Repeat</td>
</tr>
<tr>
<td>D04</td>
<td>Nonexistent Alarm</td>
</tr>
<tr>
<td></td>
<td><strong>Equation Library Messages</strong></td>
</tr>
<tr>
<td>E303</td>
<td>Keyword Conflict</td>
</tr>
<tr>
<td>E304</td>
<td>No Picture Available</td>
</tr>
<tr>
<td></td>
<td><strong>Multiple-Equation Solver Messages</strong></td>
</tr>
<tr>
<td>E401</td>
<td>Invalid Mpar</td>
</tr>
<tr>
<td>E402</td>
<td>Single Equation</td>
</tr>
<tr>
<td>E403</td>
<td>EQ Invalid for MINIT</td>
</tr>
<tr>
<td>E404</td>
<td>Too Many Unknowns</td>
</tr>
<tr>
<td>E405</td>
<td>All Variables Known</td>
</tr>
<tr>
<td>E406</td>
<td>Illegal During MROOT</td>
</tr>
<tr>
<td></td>
<td><strong>Miscellaneous Messages</strong></td>
</tr>
<tr>
<td>70000</td>
<td>(user-defined message created with DOERR)</td>
</tr>
</tbody>
</table>
# Table of Units

## HP 48 Units

<table>
<thead>
<tr>
<th>Unit (Full Name)</th>
<th>Value in SI Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (are)</td>
<td>100 m²</td>
</tr>
<tr>
<td>A (ampere)</td>
<td>1 A</td>
</tr>
<tr>
<td>acre (acre)</td>
<td>4046.87260987 m²</td>
</tr>
<tr>
<td>arcmin (minute of arc)</td>
<td>2.90888208666 x 10⁻⁴ r</td>
</tr>
<tr>
<td>arcs (second of arc)</td>
<td>4.8481368111 x 10⁻⁶ r</td>
</tr>
<tr>
<td>atm (atmosphere)</td>
<td>101325 kg/m·s²</td>
</tr>
<tr>
<td>au (astronomical unit)</td>
<td>1.495979 x 10¹¹ m</td>
</tr>
<tr>
<td>Å (Angstrom)</td>
<td>1 x 10⁻¹⁰ m</td>
</tr>
<tr>
<td>b (barn)</td>
<td>1 x 10⁻²⁸ m²</td>
</tr>
<tr>
<td>bar (bar)</td>
<td>100000 kg/m·s²</td>
</tr>
<tr>
<td>bbl (barrel)</td>
<td>0.158987294928 m³</td>
</tr>
<tr>
<td>Bq (becquerel)</td>
<td>1 1/s</td>
</tr>
<tr>
<td>Btu (international table Btu)</td>
<td>1055.05585262 kg·m²/s²</td>
</tr>
<tr>
<td>bu (bushel)</td>
<td>0.03523907 m³</td>
</tr>
<tr>
<td>°C (degree Celsius)</td>
<td>1 K or 274.15 K</td>
</tr>
<tr>
<td>c (speed of light)</td>
<td>299792458 m/s</td>
</tr>
<tr>
<td>C (coulomb)</td>
<td>1 A·s</td>
</tr>
<tr>
<td>cal (calorie)</td>
<td>4.1868 kg·m²/s²</td>
</tr>
<tr>
<td>cd (candela)</td>
<td>1 cd</td>
</tr>
<tr>
<td>chain (chain)</td>
<td>20.1168402337 m</td>
</tr>
<tr>
<td>Ci (curie)</td>
<td>3.7 x 10¹⁰ 1/s</td>
</tr>
<tr>
<td>ct (carat)</td>
<td>.0002 kg</td>
</tr>
<tr>
<td>cu (US cup)</td>
<td>2.365882365 x 10⁻⁴ m³</td>
</tr>
<tr>
<td>° (degree)</td>
<td>1.74532925199 x 10⁻² r</td>
</tr>
<tr>
<td>d (day)</td>
<td>86400 s</td>
</tr>
</tbody>
</table>

Table of Units  B-1
<table>
<thead>
<tr>
<th>Unit (Full Name)</th>
<th>Value in SI Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>dyn (dyne)</td>
<td>0.00001 kg·m/s²</td>
</tr>
<tr>
<td>erg (erg)</td>
<td>0.0000001 kg·m²/s²</td>
</tr>
<tr>
<td>eV (electron volt)</td>
<td>1.60217733 × 10⁻¹⁹ kg·m²/s²</td>
</tr>
<tr>
<td>F (farad)</td>
<td>1 A²·s⁴/kg·m²</td>
</tr>
<tr>
<td>°F (degrees Fahrenheit)</td>
<td>0.555555555556 K or 255.927777778 K</td>
</tr>
<tr>
<td>fath (fathom)</td>
<td>1.82880365761 m</td>
</tr>
<tr>
<td>fbm (board foot)</td>
<td>0.002359737216 m³</td>
</tr>
<tr>
<td>fc (footcandle)</td>
<td>10.7639104167 cd·sr/m²</td>
</tr>
<tr>
<td>Fdy (faraday)</td>
<td>96487 A·s</td>
</tr>
<tr>
<td>fermi (fermi)</td>
<td>1 × 10⁻¹⁵ m</td>
</tr>
<tr>
<td>flam (footlambert)</td>
<td>3.42625909964 cd/m²</td>
</tr>
<tr>
<td>ft (international foot)</td>
<td>0.3048 m</td>
</tr>
<tr>
<td>ftUS (survey foot)</td>
<td>0.304800609601 m</td>
</tr>
<tr>
<td>g (gram)</td>
<td>0.001 kg</td>
</tr>
<tr>
<td>ga (standard freefall)</td>
<td>9.80665 m/s²</td>
</tr>
<tr>
<td>gal (US gallon)</td>
<td>0.003785411784 m³</td>
</tr>
<tr>
<td>galC (Canadian gallon)</td>
<td>0.00454609 m³</td>
</tr>
<tr>
<td>galUK (UK gallon)</td>
<td>0.004546092 m³</td>
</tr>
<tr>
<td>gf (gram-force)</td>
<td>0.00980665 kg·m/s²</td>
</tr>
<tr>
<td>grad (gradient)</td>
<td>1.57079632679 × 10⁻² r</td>
</tr>
<tr>
<td>grain (grain)</td>
<td>0.00006479891 kg</td>
</tr>
<tr>
<td>Gy (gray)</td>
<td>1 m²/s²</td>
</tr>
<tr>
<td>H (henry)</td>
<td>1 kg·m²/A²·s²</td>
</tr>
<tr>
<td>h (Hour)</td>
<td>3600 s</td>
</tr>
<tr>
<td>hp (horsepower)</td>
<td>745.699871582 kg·m²/s³</td>
</tr>
<tr>
<td>Hz (hertz)</td>
<td>1/s</td>
</tr>
<tr>
<td>in (inch)</td>
<td>0.0254 m</td>
</tr>
<tr>
<td>inHg (inches of mercury, 0°C)</td>
<td>3386.38815789 kg·m²/s²</td>
</tr>
<tr>
<td>inH20 (inches of water, 60°F)</td>
<td>248.84 kg/m·s²</td>
</tr>
<tr>
<td>J (joule)</td>
<td>1 kg·m²/s²</td>
</tr>
<tr>
<td>K (kelvins)</td>
<td>1 K</td>
</tr>
<tr>
<td>kg (kilogram)</td>
<td>1 kg</td>
</tr>
<tr>
<td>Unit (Full Name)</td>
<td>Value in SI Units</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>kip (kilopound-force)</td>
<td>4448.22161526 kg·m/s²</td>
</tr>
<tr>
<td>knot (nautical miles per hour)</td>
<td>.514444444444 m/s</td>
</tr>
<tr>
<td>kph (kilometers per hour)</td>
<td>.277777777778 m/s</td>
</tr>
<tr>
<td>l (liter)</td>
<td>.001 m³</td>
</tr>
<tr>
<td>lam (lambert)</td>
<td>3183.09886184 cd/m²</td>
</tr>
<tr>
<td>lb (avoirdupois pound)</td>
<td>.45359237 kg</td>
</tr>
<tr>
<td>lbf (pound-force)</td>
<td>4.44822161526 kg·m/s²</td>
</tr>
<tr>
<td>lbt (troy pound)</td>
<td>.3732417216 kg</td>
</tr>
<tr>
<td>lm (lumen)</td>
<td>1 cd·sr</td>
</tr>
<tr>
<td>lx (lux)</td>
<td>1 cd·sr/m²</td>
</tr>
<tr>
<td>lyr (light year)</td>
<td>9.46052840488 x 10¹⁵ m</td>
</tr>
<tr>
<td>m (meter)</td>
<td>1 m</td>
</tr>
<tr>
<td>μ (micron)</td>
<td>1 x 10⁻⁶ m</td>
</tr>
<tr>
<td>mho (mho)</td>
<td>1 A²·s³/kg·m²</td>
</tr>
<tr>
<td>mi (international mile)</td>
<td>1609.344 m</td>
</tr>
<tr>
<td>mil (mil)</td>
<td>.0000254 m</td>
</tr>
<tr>
<td>min (minute)</td>
<td>60 s</td>
</tr>
<tr>
<td>miUS (US statute mile)</td>
<td>1609.34721869 m</td>
</tr>
<tr>
<td>mmHg (millimeter of mercury (torr), 0°C)</td>
<td>133.322368421 kg/m·s²</td>
</tr>
<tr>
<td>mol (mole)</td>
<td>1 mol</td>
</tr>
<tr>
<td>mph (miles per hour)</td>
<td>.44704 m/s</td>
</tr>
<tr>
<td>N (newton)</td>
<td>1 kg·m/s²</td>
</tr>
<tr>
<td>nmi (nautical mile)</td>
<td>1852 m</td>
</tr>
<tr>
<td>Ω (ohm)</td>
<td>1 kg·m²/A²·s³</td>
</tr>
<tr>
<td>oz (ounce)</td>
<td>.028349523125 kg</td>
</tr>
<tr>
<td>ozf1 (US fluid ounce)</td>
<td>2.95735295625 x 10⁻⁵ m³</td>
</tr>
<tr>
<td>ozt (troy ounce)</td>
<td>.0311034768 kg</td>
</tr>
<tr>
<td>ozUK (UK fluid ounce)</td>
<td>2.8413075 x 10⁻⁵ m³</td>
</tr>
<tr>
<td>P (poise)</td>
<td>.1 kg/m·s</td>
</tr>
<tr>
<td>Pa (pascal)</td>
<td>1 kg/m·s²</td>
</tr>
<tr>
<td>pc (parsec)</td>
<td>3.08567818585 x 10¹⁶ m</td>
</tr>
<tr>
<td>pd1 (poundal)</td>
<td>.138254954376 kg·m/s²</td>
</tr>
<tr>
<td>ph (phot)</td>
<td>10000 cd·sr/m²</td>
</tr>
<tr>
<td>pk (peck)</td>
<td>.0088097675 m³</td>
</tr>
<tr>
<td>Unit (Full Name)</td>
<td>Value in SI Units</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>psi (pounds per square inch)</td>
<td>6894.75729317 kg/m·s²</td>
</tr>
<tr>
<td>pt (pint)</td>
<td>.000473176473 m³</td>
</tr>
<tr>
<td>qt (quart)</td>
<td>.000946352946 m³</td>
</tr>
<tr>
<td>r (radian)</td>
<td>1 r</td>
</tr>
<tr>
<td>R (roentgen)</td>
<td>.000258 A·s/kg</td>
</tr>
<tr>
<td>°R (degrees Rankine)</td>
<td>0.5555555555556 K</td>
</tr>
<tr>
<td>rad (rad)</td>
<td>.01 m²/s²</td>
</tr>
<tr>
<td>rd (rod)</td>
<td>5.02921005842 m</td>
</tr>
<tr>
<td>rem (rem)</td>
<td>.01 m²/s²</td>
</tr>
<tr>
<td>s (second)</td>
<td>1 s</td>
</tr>
<tr>
<td>S (siemens)</td>
<td>1 A²·s³/kg·m²</td>
</tr>
<tr>
<td>sb (stilb)</td>
<td>10000 cd/m²</td>
</tr>
<tr>
<td>slug (slug)</td>
<td>14.5939029372 kg</td>
</tr>
<tr>
<td>sr (steradian)</td>
<td>1 sr</td>
</tr>
<tr>
<td>st (stere)</td>
<td>1 m³</td>
</tr>
<tr>
<td>St (stokes)</td>
<td>.0001 m²/s</td>
</tr>
<tr>
<td>Sv (sievert)</td>
<td>1 m²/s²</td>
</tr>
<tr>
<td>t (metric ton)</td>
<td>1000 kg</td>
</tr>
<tr>
<td>T (tesla)</td>
<td>1 kg/A·s²</td>
</tr>
<tr>
<td>tbsp (tablespoon)</td>
<td>1.47867647813 x 10⁻⁵ m³</td>
</tr>
<tr>
<td>therm (EEC therm)</td>
<td>105506000 kg/m²/s²</td>
</tr>
<tr>
<td>ton (short ton)</td>
<td>907.18474 kg</td>
</tr>
<tr>
<td>tonUK (long ton (UK))</td>
<td>1016.0469088 kg</td>
</tr>
<tr>
<td>torr (torr (mmHg))</td>
<td>133.322368421 kg/m·s²</td>
</tr>
<tr>
<td>tsp (teaspoon)</td>
<td>4.92892159375 x 10⁻⁶ m³</td>
</tr>
<tr>
<td>u (unified atomic mass)</td>
<td>1.6605402 x 10⁻²⁷ kg</td>
</tr>
<tr>
<td>V (volt)</td>
<td>1 kg·m²/A·s³</td>
</tr>
<tr>
<td>W (watt)</td>
<td>1 kg·m²/s³</td>
</tr>
<tr>
<td>Wb (weber)</td>
<td>1 kg·m²/A·s²</td>
</tr>
<tr>
<td>yr (international yard)</td>
<td>.9144 m</td>
</tr>
<tr>
<td>yr (year)</td>
<td>31556925.9747 s</td>
</tr>
</tbody>
</table>
System Flags

This appendix lists the HP 48 system flags. You can set, clear, and test all flags. The default state of the flags is clear—except for the Binary Integer Wordsize flags (flags −5 through −10).

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
</table>
| −1   | Principal Solution.  
      | *Clear*: QUAD and ISOL return a result representing all possible solutions.  
      | *Set*: QUAD and ISOL return only the principal solution. |
| −2   | Symbolic Constants.  
      | *Clear*: Symbolic constants (e, i, π, MAXR, and MINR) retain their symbolic form when evaluated, unless the Numerical Results flag −3 is set.  
      | *Set*: Symbolic constants evaluate to numbers, regardless of the state of the Numerical Results flag −3. |
| −3   | Numerical Results.  
      | *Clear*: Functions with symbolic arguments, including symbolic constants, evaluate to symbolic results.  
      | *Set*: Functions with symbolic arguments, including symbolic constants, evaluate to numbers. |
| −4   | Not used. |
| −5   | Binary Integer Wordsize.  
<pre><code>  | thru Combined states of flags −5 through −10 set the wordsize from 1 to 64 bits. |
</code></pre>
<p>| −10  |             |</p>
<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-13</td>
<td>Not used.</td>
</tr>
<tr>
<td>-17 and -18</td>
<td>Degrees: -17 clear, -18 clear. and Radians: -17 set.</td>
</tr>
<tr>
<td>-17 and -18</td>
<td>Grads: -17 clear, -18 set.</td>
</tr>
<tr>
<td>-19</td>
<td>Clear: →V2 and creates a 2-dimensional vector from 2 real numbers. Set: →V2 and creates a complex number from 2 real numbers.</td>
</tr>
<tr>
<td>-23</td>
<td>Negative Underflow Indicator.</td>
</tr>
<tr>
<td>-24</td>
<td>Positive Underflow Indicator.</td>
</tr>
<tr>
<td>-25</td>
<td>Overflow Indicator.</td>
</tr>
<tr>
<td>-26</td>
<td>Infinite Result Indicator. When an exception occurs, corresponding flag (-23 through -26) is set only if the exception is not treated as an error.</td>
</tr>
<tr>
<td>Flag</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| -27   | Display of symbolic complex numbers.  
         *Clear:* Displays symbolic complex numbers in coordinate form (i.e. \((x, y)\)).  
         *Set:* Displays symbolic complex numbers using \(i\) (i.e. \(x+yi\)). |
| -28   | Simultaneous Plotting of Multiple Functions.  
         *Clear:* Multiple equations are plotted serially.  
         *Set:* Multiple equations are plotted simultaneously. |
| -29   | Draw Axes.  
         *Clear:* Axes are drawn for two-dimensional and statistical plots.  
         *Set:* Axes are not drawn for two-dimensional and statistical plots. |
| -30   | Not used. |
| -31   | Curve Filling.  
         *Clear:* Curve filling between plotted points enabled.  
         *Set:* Curve filling between plotted points suppressed. |
| -32   | Graphics Cursor.  
         *Clear:* Graphics cursor always dark.  
         *Set:* Graphics cursor dark on light background and light on dark background. |
| -33   | I/O Device.  
         *Clear:* I/O directed to serial port.  
         *Set:* I/O directed to IR port. |
| -34   | Printing Device.  
         *Clear:* Printer output directed to IR printer.  
         *Set:* Printer output directed to serial port if flag -33 is clear. |
| -35   | I/O Data Format.  
         *Clear:* Objects transmitted in ASCII form.  
         *Set:* Objects transmitted in binary (memory image) form. |
<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
</table>
| -36  | I/O Receive Overwrite.  
      *Clear*: If file name received by HP 48 matches existing HP 48 variable name, new variable name with number extension is created to prevent overwrite.  
      *Set*: If file name received by HP 48 matches existing HP 48 variable name, existing variable is overwritten. |
| -37  | Double-Spaced Printing.  
      *Clear*: Single-spaced printing.  
      *Set*: Double-spaced printing. |
| -38  | Line Feed.  
      *Clear*: Linefeed added at end of each print line.  
      *Set*: No linefeed added at end of each print line. |
| -39  | I/O Messages.  
      *Clear*: I/O messages displayed.  
      *Set*: I/O messages suppressed. |
| -40  | Clock Display.  
      *Clear*: Clock displayed only when TIME menu selected.  
      *Set*: Ticking clock displayed at all times. |
| -41  | Clock Format.  
      *Clear*: 12-hour clock.  
      *Set*: 24-hour clock. |
| -42  | Date Format.  
      *Clear*: MM/DD/YY (month/day/year) format.  
      *Set*: DD.MM.YY (day.month.year) format. |
| -43  | Repeat Alarms Not Rescheduled.  
      *Clear*: Unacknowledged repeat appointment alarms automatically rescheduled.  
      *Set*: Unacknowledged repeat appointment alarms not rescheduled. |
| -44  | Acknowledged Alarms Saved.  
      *Clear*: Acknowledged appointment alarms deleted from alarm list.  
      *Set*: Acknowledged appointment alarms saved in alarm list. |
<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-45 thru 48</td>
<td>Number of Decimal Digits. Combined states of flags -45 through -48 sets number of decimal digits in Fix, Scientific, and Engineering modes.</td>
</tr>
<tr>
<td>-52</td>
<td>Single-Line Display. Clear: Display gives preference to object in level 1, using up to four lines of stack display. Set: Display of object in level 1 restricted to one line.</td>
</tr>
<tr>
<td>-54</td>
<td>Tiny Array Elements. Clear: Singular values computed by RANK (and other commands that compute the rank of a matrix) that are more than $1 \times 10^{-14}$ times smaller than the largest computed singular value in the matrix are converted to zero. Automatic rounding for DET is enabled. Set: Small computed singular values (see above) not converted. Automatic rounding for DET is disabled.</td>
</tr>
<tr>
<td>Flag</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| -57  | Alarm Beep.  
      | *Clear:* Alarm beep enabled.  
      | *Set:* Alarm beep suppressed. |
| -58  | Verbose Messages.  
      | *Clear:* Parameter variable data automatically displayed.  
      | *Set:* Automatic display of parameter variable data is suppressed. |
| -59  | Fast Browser Display.  
      | *Clear:* Variable Browser shows variable names and contents.  
      | *Set:* Variable Browser shows variable names only. |
| -60  | Alpha Lock.  
      | *Clear:* Single-Alpha activated by pressing $\alpha$ once. Alpha lock activated by pressing $\alpha$ twice.  
      | *Set:* Alpha lock activated by pressing $\alpha$ once. (Single-Alpha not available.) |
| -61  | User-Mode Lock.  
      | *Clear:* 1-User mode activated by pressing $\leftarrow\text{USER}$ once. User mode activated by pressing $\leftarrow\text{USER}$ twice.  
      | *Set:* User mode activated by pressing $\leftarrow\text{USER}$ once. (1-User mode not available.) |
| -62  | User Mode.  
      | *Clear:* User mode not active.  
      | *Set:* User mode active. |
| -63  | Vectored (ENTER).  
      | *Clear:* (ENTER) evaluates command line.  
      | *Set:* User-defined (ENTER) activated. |
| -64  | Index Wrap Indicator.  
      | *Clear:* Last execution of GETI or PUTI did not increment index to first element.  
      | *Set:* Last execution of GETI or PUTI did increment index to first element. |
Reserved Variables

The HP 48 uses the following reserved variables. These have specific purposes, and their names are used as implicit arguments for certain commands. Avoid using these variables’ names for other purposes, or you may interfere with the execution of the commands that use these variables.

You can change some of the values in these variables with programmable commands, while others require you to store new values into the appropriate place.

<table>
<thead>
<tr>
<th>Reserved Variable</th>
<th>What It Contains</th>
<th>Used By</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALRMDAT</td>
<td>Alarm parameters.</td>
<td>TIME ALRM operations</td>
</tr>
<tr>
<td>CST</td>
<td>List defining the CST (custom) menu.</td>
<td>MENU, [CST]</td>
</tr>
<tr>
<td>“der”-names</td>
<td>User-defined derivative.</td>
<td>∂</td>
</tr>
<tr>
<td>EQ</td>
<td>Current equation.</td>
<td>ROOT, DRAW</td>
</tr>
<tr>
<td>EXPR</td>
<td>Current expression.</td>
<td>SYMBOLIC</td>
</tr>
<tr>
<td>IOPAR</td>
<td>I/O parameters.</td>
<td>I/O commands</td>
</tr>
<tr>
<td>MHpar</td>
<td>Minehunt game status.</td>
<td>MINEHUNT</td>
</tr>
<tr>
<td>Mpar</td>
<td>Multiple-Equation Solver equations.</td>
<td>EQ LIB</td>
</tr>
<tr>
<td>n1, n2, ...</td>
<td>Arbitrary integers.</td>
<td>ISOL, QUAD</td>
</tr>
<tr>
<td>Nmines</td>
<td>Minehunt game data.</td>
<td>MINEHUNT</td>
</tr>
</tbody>
</table>
Reserved Variable | What It Contains | Used By
---|---|---
PPAR | Plotting parameters. | DRAW
PRTPAR | Printing parameters. | PRINT commands
s1, s2, ... | Arbitrary signs. | ISOL, QUAD
VPAR | Viewing parameters. | DRAW
ZPAR | Plot zoom factors. | DRAW
ΣDAT | Statistical data. | Statistics application, DRAW
ΣPAR | Statistical parameters. | Statistics application, DRAW

**Contents of the Reserved Variables**

Most reserved variables (except ALRMDAT, IOPAR and PRTPAR) can be stored with different contents in different directories. This allows you, for example, to save several sets of statistical data in different directories.

**ALRMDAT**

ALRMDAT does not reside in a particular directory. You cannot access the variable itself, but you can access its data from any directory using the RCLALARM and STOALARM commands, or through the Alarm Catalog.

ALRMDAT contains a list of these alarm parameters:
<table>
<thead>
<tr>
<th>Parameter (Command)</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>date</code> (→DATE)</td>
<td>A real number specifying the date of the alarm: <code>MM.DDYYYY</code> (or <code>DD.MMYYYY</code> if flag -42 is set). If <code>YYYY</code> is not included, the current year is used.</td>
<td>Current date.</td>
</tr>
<tr>
<td><code>time</code> (→TIME)</td>
<td>A real number specifying the time of the alarm: <code>HH.MMSS</code>.</td>
<td>00.0000</td>
</tr>
</tbody>
</table>
| `action`            | A string or object:  
  - a string creates an *appointment alarm*, which beeps and displays the string  
  - any other object creates a *control alarm*, which executes the object | Empty string (appointment alarm). |
| `repeat`            | A real number specifying the interval between automatic recurrences of the alarm, given in ticks (a tick is $\frac{1}{8192}$ of a second). | 0 |

Parameters without commands can be modified with a program by storing new values in the list contained in `ALRMDAT` (use the PUT command).

**CST**

*CST* contains a list (or a name specifying a list) of the objects that define the CST (*custom*) menu. Objects in the custom menu behave as do objects in built-in menus. For example:

- Names behave like the VAR menu keys. Thus, if `ABC` is a variable name, `ABC` evaluates `ABC`, `ABC` recalls its contents, and `ABC` stores new contents in `ABC`.

- The menu label for the name of a directory has a bar over the left side of the label; pressing the menu key switches to that directory.

- Unit objects act like unit catalog entries (and have left-shifted conversion capabilities, for example).
String keys echo the string.

You can include backup objects in the list defining a custom menu by tagging the name of the backup object with its port location (0 through 33).

You can specify menu labels and key actions independently by replacing a single object within the custom-menu list with a list of the form "label-object action-object". (See “Customizing Menus” and “Enhancing Custom Menus” in chapter 30 of the HP 48 User’s Guide for more information.)

To provide different shifted actions for custom menu keys, action-object can be a list containing three action objects in this order:

- The unshifted action (required if you want to specify the shifted actions).
- The left-shifted action.
- The right-shifted action.


“der-” Names

If \(\partial\) is applied to a function for which there is no built-in derivative, it returns a new function whose name is “der” followed by the original function name. These “der”-function names are reserved variable names.

For an example, refer to “Creating User-Defined Derivatives” in chapter 20 of the HP 48 User’s Guide.

EQ

EQ contains the current equation or the name of the variable containing the current equation.

EQ supplies the equation for ROOT, as well as for the plotting command DRAW when the plot type is FUNCTION, CONIC, POLAR, PARAMETER, TRUTH, or DIFFEQ. (\(\Sigma\)DAT supplies the information when the plot type is HISTOGRAM, BAR, or SCATTER.)
The object in $EQ$ can be an algebraic object, a number, a name, or a program. How DRAW interprets $EQ$ depends on the plot type.

For graphics use, $EQ$ can also be a list of equations or other objects. If $EQ$ contains a list, then DRAW treats each object in turn as the current equation, and plots them successively. However, ROOT in the HP Solve application cannot solve an $EQ$ containing a list.

To alter the contents of $EQ$, use the command STEQ.

**EXPR**

$EXPR$ contains the current algebraic expression (or the name of the variable containing the current expression) used by the SYMBOLIC application and its associated commands. The object in $EQ$ must be an algebraic or a name.

**IOPAR**

$IOPAR$ is a variable in the $HOME$ directory that contains a list of the I/O parameters needed for a communications link with a computer. It is created the first time you transfer data or open the serial port (OPENIO), and is automatically updated whenever you change the I/O settings. All $IOPAR$ parameters are integers.

<table>
<thead>
<tr>
<th>Parameter (Command)</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>baud (BAUD)</td>
<td>The baud rate: 1200, 2400, 4800, or 9600. The parity used: 0=none, 1=odd, 2=even, 3=mark, 4=space. The value can be positive or negative: a positive parity is used upon both transmit and receive; a negative parity is used only upon transmit.</td>
<td>9600</td>
</tr>
<tr>
<td>parity (PARITY)</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Parameter (Command)</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>receive pacing</td>
<td>Controls whether receive pacing is used: a nonzero real value enables pacing, while zero disables it. Receive pacing sends an XOFF signal when the receive buffer is almost full, and sends an XON signal when it can take more data again. Pacing is not used for Kermit I/O, but is used for other serial I/O transfers.</td>
<td>0 (no pacing)</td>
</tr>
<tr>
<td>transmit pacing</td>
<td>Controls whether transmit pacing is used: a nonzero real value enables pacing, while zero disables it. Transmit pacing stops transmission upon receipt of XOFF, and resumes transmission upon receipt of XON. Pacing is not used for Kermit I/O, but is used for other serial I/O transfers.</td>
<td>0 (no pacing)</td>
</tr>
<tr>
<td>checksum (CKSM)</td>
<td>Error-detection scheme requested when initiating SEND: - 1=1-digit arithmetic checksum - 2=2-digit arithmetic checksum - 3=3-digit cyclic redundancy check.</td>
<td>3</td>
</tr>
<tr>
<td>translation code (TRANSIO)</td>
<td>Controls which characters are translated: - 0=none - 1=translate character 10 (line feed only) to/from characters 10 and 13 (line feed and carriage return) - 2=translate characters with numbers 128 through 159 (80–9F hex) - 3=translate characters with numbers 128 through 255.</td>
<td>1</td>
</tr>
</tbody>
</table>

Parameters without commands can be modified with a program by storing new values in the list contained in IOPAR (use the PUT command), or by editing IOPAR directly.
MHpar

MHpar stores the status of an interrupted Minehunt game. MHpar is created when you exit Minehunt by pressing (STO). If MHpar still exists when you restart Minehunt, the interrupted game resumes and MHpar is purged.

Mpar

Mpar is created when you use the Equation Library’s Multiple-Equation Solver, and it stores the set of equations you’re using.

When the Equation Library starts the Multiple-Equation Solver, it first stores a list of the equation set in EQ, and stores the equation set, a list of variables, and additional information in Mpar. Mpar is then used to set up the Solver menu for the current equation set.

Mpar is structured as library data dedicated to the Multiple Equation Solver application. This means that although you can view and edit Mpar directly, you can edit it only indirectly by executing commands that modify it.

You can also use the MINIT command (EQ LIB MES MINIT) to create MHpar from a set of equations on the stack. See “Defining a Set of Equations” in chapter 25 of the HP 48 User’s Guide.

n1, n2, …

The ISOL and QUAD commands return general solutions (as opposed to principal solutions) for operations. A general solution contains variables for arbitrary integers or arbitrary signs, or both.

The variable n1 represents an arbitrary integer 0, ±1, ±2, etc. Additional arbitrary integers are represented by n2, n3, etc.

If flag -1 is set, then ISOL and QUAD return principal solutions, in which case the arbitrary integer is always zero.
Nmines

Nmines is a variable you create in the current directory to control the number of mines used in the Minehunt game. Nmines contains an integer in the range 1 to 64; if Nmines is negative, the mines are visible during the game.

PPAR

PPAR is a variable in the current directory. It contains a list of plotting parameters used by the DRAW command for all mathematical and statistical plots, by AUTO for autoscaling, and by the interactive (nonprogrammable) graphics operations.

<table>
<thead>
<tr>
<th>Parameter (Command)</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x_{\text{min}}, y_{\text{min}})$ (XRNG, YRNG)</td>
<td>A complex number specifying the lower left corner of PICT (the lower left corner of the display range).</td>
<td>$(−6.50, −3.1)$</td>
</tr>
<tr>
<td>$(x_{\text{max}}, y_{\text{max}})$ (XRNG, YRNG)</td>
<td>A complex number specifying the upper right corner of PICT (the upper right corner of the display range).</td>
<td>$(6.5, 3.2)$</td>
</tr>
<tr>
<td>indep (INDEP)</td>
<td>A name specifying the independent variable, or a list containing that name and two numbers that specify the minimum and maximum values for the independent variable (the plotting range).</td>
<td>$X$</td>
</tr>
<tr>
<td>Parameter (Command)</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>res</code> (RES)</td>
<td>Resolution. A real number specifying the interval between values of the independent variable. For plots of equations, this determines the plotting interval along the x-axis. A binary number specifies the pixel resolution (how many columns of pixels between points). An integer specifies the resolution in user units (how many user units between points). Resolution for statistical plots is different (see below).</td>
<td>0</td>
</tr>
</tbody>
</table>
| `axes` (AXES)       | A complex number specifying the user-unit coordinates of the plot origin, or a list containing the following:  
  - the complex number specifying the origin  
  - a real number, binary integer, or list containing two real numbers or binary integers specifying the tick-mark annotation (see ATICK)  
  - two strings specifying labels for the horizontal and vertical axes | (0, 0) |
<p>| <code>ptype</code> (BAR, etc.) | A command name specifying the plot type (BAR, CONIC, DIFSEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, or YSLICE). | FUNCTION |</p>
<table>
<thead>
<tr>
<th>Parameter (Command)</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>depend (DEPND)</td>
<td>A name specifying the dependent variable, or a list containing the name and two numbers that specify vertical plotting range. For DIFFEQ, the second element of the list may also be a real vector that represents the initial value.</td>
<td>Y</td>
</tr>
</tbody>
</table>

Parameters without commands can be modified with a program by storing new values in the list contained in PPAR (use the PUT command).

The RESET operation (PLOT PPAR RESET) resets the PPAR parameters (except ptype) to their default values, and erases PICT.

Note that res behaves differently for the statistical plot types BAR, HISTOGRAM, and SCATTER than for other plot types. For BAR, res specifies bar width; for HISTOGRAM, res specifies bin width; res does not affect SCATTER.
**PRTPAR**

*PRTPAR* is a variable in the *HOME* directory that contains a list of printing parameters. It is created automatically the first time you use a printing command.

<table>
<thead>
<tr>
<th>Parameter (Command)</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay time (DELAY)</td>
<td>A real number, in the range 0 to 6.9, specifying the number of seconds the HP 48 waits between sending lines. This should be at least as long as the time required to print the longest line. If the delay is too short for the printer, you will lose data. The delay setting also affects serial printing if transmit-pacing (in IOPAR) is not being used.</td>
<td>1.8</td>
</tr>
<tr>
<td>remap (OLDPRT stores the character-remapping string for the HP 82240A Infrared Printer)</td>
<td>A string defining the current remapping of the extended character set for printing. The string can contain as many characters as you want to remap, with the first character being the new character 128, the second being the new character 129, etc. (Any character number that exceeds the string length will not be remapped.) See the example below.</td>
<td>Empty string.</td>
</tr>
<tr>
<td>line length</td>
<td>A real number specifying the number of characters in a line for serial printing. This does not affect infrared printing.</td>
<td>80</td>
</tr>
<tr>
<td>line termination</td>
<td>A string specifying the line-termination method for serial printing. This does not affect infrared printing.</td>
<td>Control characters 13 (carriage return) and 10 (line feed).</td>
</tr>
</tbody>
</table>
Parameters without commands can be modified with a program by storing new values in the list contained in \texttt{PRTPAR} (use the PUT command).

A change in a parameter is effective immediately, \textit{except} when printing the display using the simultaneous keystrokes \texttt{(ON)(I/O)} (because this does not use \texttt{PRTPAR}). This printing method is affected only by the delay parameter, a change in which will not affect \texttt{(ON)(I/O)} until after the next printing command has been executed. To use a new delay time with \texttt{(ON)(I/O)} immediately, use the \texttt{DELAY} command.

\textbf{Example:} If the remapping string were “ABCDEFGH” and the character to be printed had value 131, then the character actually printed would be “D”, since 131—128=3 and “A” has the value zero. A character code of 136 or greater would not be remapped since 136—128=8, which exceeds the length of the string.

\textbf{s1, s2, \ldots}

The \texttt{ISOL} and \texttt{QUAD} commands return \textit{general} solutions (as opposed to \textit{principal} solutions) for operations. A general solution contains variables for arbitrary integers or arbitrary signs or both.

The variable \texttt{s1} represents an arbitrary + or − sign. Additional arbitrary signs are represented by \texttt{s2}, \texttt{s3}, etc.

If flag −1 is set, then \texttt{ISOL} and \texttt{QUAD} return principal solutions, in which case the arbitrary sign is always +1.
VPAR

VPAR is a variable in the current directory. It contains a list of parameters used by the 3D plot types. The main data structure stored in VPAR describes the “view volume,” the abstract three-dimensional region in which the function is plotted.

<table>
<thead>
<tr>
<th>Parameter (Command)</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x_{\text{left}}, x_{\text{right}})) (\text{XVOL})</td>
<td>Real numbers that specify the width of the view volume.</td>
<td>((-1, 1))</td>
</tr>
<tr>
<td>((y_{\text{far}}, y_{\text{near}})) (\text{YVOL})</td>
<td>Real numbers that specify the depth of the view volume.</td>
<td>((-1, 1))</td>
</tr>
<tr>
<td>((z_{\text{low}}, z_{\text{high}})) (\text{ZVOL})</td>
<td>Real numbers that specify the height of the view volume.</td>
<td>((-1, 1))</td>
</tr>
<tr>
<td>((x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}})) (\text{EYEPT})</td>
<td>Real numbers that specify the point in space from which the plot is viewed.</td>
<td>((0, -3, 0))</td>
</tr>
<tr>
<td>((x_{\text{step}}, y_{\text{step}})) (\text{NUMX,NUMY})</td>
<td>Real numbers that specify the increments between of x-coordinates and y-coordinates plotted. The increments are equal to the range for the axes divided by the number of steps. Used instead of (or in combination with) res.</td>
<td>((10, 8))</td>
</tr>
<tr>
<td>((x_{x_{\text{left}}, x_{x_{\text{right}}}})) (\text{XXRNG})</td>
<td>Real numbers that specify the width of the input plane (domain). Used by GRIDMAP and PARSURFACE.</td>
<td>((-1, 1))</td>
</tr>
<tr>
<td>((y_{y_{\text{far}}, y_{y_{\text{near}}}})) (\text{YYRNG})</td>
<td>Real numbers that specify the depth of the input plane (domain). Used by GRIDMAP and PARSURFACE.</td>
<td>((-1, 1))</td>
</tr>
</tbody>
</table>

Parameters without commands can be modified programmatically by storing new values in the list contained in VPAR (use the PUT command).

The \texttt{RESET} operation \(\text{RESET PLOT NXT 3D VPAR NXT} \text{RESET}\) resets the VPAR parameters to their default values.
**ZPAR**

*ZPAR* is a variable in the current directory. It contains a list of zooming parameters used by the DRAW command for all 2-D mathematical and statistical plots.

<table>
<thead>
<tr>
<th>Parameter (Command)</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h\text{-factor} )</td>
<td>Real number that specifies the horizontal zoom factor.</td>
<td>4</td>
</tr>
<tr>
<td>( v\text{-factor} )</td>
<td>Real number that specifies the vertical zoom factor.</td>
<td>4</td>
</tr>
<tr>
<td>( \text{recenter flag} )</td>
<td>0 or 1 depending on whether the recenter at crosshairs option was selected in the set zoom factors input form.</td>
<td>0</td>
</tr>
<tr>
<td>{ list }</td>
<td>An empty list, or a copy of the last ( PPAR ).</td>
<td></td>
</tr>
</tbody>
</table>

Use the set zoom factors input form (\([ZFACT]\)) to modify *ZPAR*. 

D-14  Reserved Variables
\section*{ΣDAT}

\textit{ΣDAT} is a variable in the current directory that contains either the current statistical matrix or the name of the variable containing this matrix. This matrix contains the data used by the Statistics applications.

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
\textit{var}_1 & \textit{var}_2 & \ldots & \textit{var}_m \\
\hline
\textit{x}_{11} & \textit{x}_{21} & \ldots & \textit{x}_{m1} \\
\textit{x}_{12} & \textit{x}_{22} & \ldots & \textit{x}_{m2} \\
\vdots & \vdots & \ddots & \vdots \\
\textit{x}_{1n} & \textit{x}_{2n} & \ldots & \textit{x}_{mn} \\
\hline
\end{tabular}
\end{center}

You can designate a new current statistical matrix by entering new data, editing the current data, or selecting another matrix.

The command \texttt{CLS} clears the current statistical matrix.
**σPAR**

σPAR is a variable in the current directory that contains either the current statistical parameter list or the name of the variable containing this list.

<table>
<thead>
<tr>
<th>Parameter (Command)</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>column\textsubscript{indep} (XCOL)</td>
<td>A real number specifying the independent-variable’s column number.</td>
<td>1</td>
</tr>
<tr>
<td>column\textsubscript{dep} (YCOL)</td>
<td>A real number specifying the dependent-variable’s column number.</td>
<td>2</td>
</tr>
<tr>
<td>intercept (LR)</td>
<td>A real number specifying the coefficient of intercept as determined by the current regression.</td>
<td>0</td>
</tr>
<tr>
<td>slope (LR)</td>
<td>A real number specifying the coefficient of slope as determined by the current regression.</td>
<td>0</td>
</tr>
<tr>
<td>model (LINFIT, etc.)</td>
<td>A command specifying the regression model (LINFIT, EXPFIT, PWRFIT, or LOGFIT).</td>
<td>LINFIT</td>
</tr>
</tbody>
</table>
New Commands

In the following tables, new commands (commands that were not available on a standard HP 48S series calculator) are arranged alphabetically and followed by brief descriptions. All of these commands are described in chapter 3.

<table>
<thead>
<tr>
<th>Command</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Adds list elements.</td>
</tr>
<tr>
<td>AMORT</td>
<td>Amortizes a loan or investment based upon the current amortization settings.</td>
</tr>
<tr>
<td>ANIMATE</td>
<td>Displays graphic objects in sequence.</td>
</tr>
<tr>
<td>ATICK</td>
<td>Sets the axes tick-mark annotation in the reserved variable PPAR.</td>
</tr>
<tr>
<td>CHOOSE</td>
<td>Creates a user-defined choose box.</td>
</tr>
<tr>
<td>CLTEACH</td>
<td>Removes the EXAMPLES subdirectory and its contents from the HOME directory.</td>
</tr>
<tr>
<td>COL+</td>
<td>Inserts an array (vector or matrix) into a matrix.</td>
</tr>
<tr>
<td>COL−</td>
<td>Deletes a column from of a matrix.</td>
</tr>
<tr>
<td>COL→</td>
<td>Transforms a series of column vectors and a column count into a matrix, or transforms a sequence of numbers and an element count into a vector.</td>
</tr>
<tr>
<td>→COL</td>
<td>Transforms a matrix into a series of column vectors, or transforms a vector into its elements.</td>
</tr>
<tr>
<td>Command</td>
<td>Brief Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>COND</td>
<td>Returns the 1-norm (column norm) condition number of a square matrix.</td>
</tr>
<tr>
<td>CONLIB</td>
<td>Opens the Constants Library catalog.</td>
</tr>
<tr>
<td>CONST</td>
<td>Returns the value of a constant.</td>
</tr>
<tr>
<td>CSWP</td>
<td>Swaps columns in a matrix.</td>
</tr>
<tr>
<td>CYLIN</td>
<td>Sets Cylindrical coordinate mode.</td>
</tr>
<tr>
<td>DARCY</td>
<td>Calculates the Darcy friction factor of certain fluid flows.</td>
</tr>
<tr>
<td>DIAG→</td>
<td>Takes an array and a specified dimension and returns a matrix whose main diagonal elements are the elements of the array.</td>
</tr>
<tr>
<td>→DIAG</td>
<td>Returns a vector that contains the major diagonal elements of a matrix.</td>
</tr>
<tr>
<td>DIFFEQ</td>
<td>Specifies differential equations as the plot type.</td>
</tr>
<tr>
<td>DOLIST</td>
<td>Applies commands, programs, or user-defined functions to lists.</td>
</tr>
<tr>
<td>DOSUBS</td>
<td>Applies a program or command to groups of elements in a list.</td>
</tr>
<tr>
<td>EGV</td>
<td>Computes the eigenvalues and right eigenvectors for a square matrix.</td>
</tr>
<tr>
<td>EGVL</td>
<td>Computes the eigenvalues of a square matrix.</td>
</tr>
<tr>
<td>ENDSUB</td>
<td>Provides a way to access the total number of sublists used while executing a program or command using DOSUBS.</td>
</tr>
<tr>
<td>EQNLIB</td>
<td>Starts the Equation Library application.</td>
</tr>
<tr>
<td>EYEPT</td>
<td>Specifies the coordinates of the eye point in a perspective plot.</td>
</tr>
<tr>
<td>Command</td>
<td>Brief Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>F0A</td>
<td>Returns the fraction of total black-body emissive power.</td>
</tr>
<tr>
<td>FANNING</td>
<td>Calculates the Fanning friction factor of certain fluid flows.</td>
</tr>
<tr>
<td>FFT</td>
<td>Computes the one- or two-dimensional discrete Fourier transform of an array.</td>
</tr>
<tr>
<td>FREE1</td>
<td>Frees the previously merged RAM in port 1.</td>
</tr>
<tr>
<td>GRIDMAP</td>
<td>Specifies grid mapping as the plot type.</td>
</tr>
<tr>
<td>HEAD</td>
<td>Returns the first element of a list or string.</td>
</tr>
<tr>
<td>IFFT</td>
<td>Computes the one- or two-dimensional inverse discrete Fourier transform of a vector or matrix.</td>
</tr>
<tr>
<td>INFORM</td>
<td>Creates a user-defined dialog box (Input Form).</td>
</tr>
<tr>
<td>LIBEVAL</td>
<td>Evaluates unnamed library objects by their memory addresses.</td>
</tr>
<tr>
<td>LININ</td>
<td>Tests whether an algebraic is structurally linear for a given variable.</td>
</tr>
<tr>
<td>ΣLIST</td>
<td>Returns the sum of the elements in a list.</td>
</tr>
<tr>
<td>ΠLIST</td>
<td>Returns the product of the elements in a list.</td>
</tr>
<tr>
<td>ΔLIST</td>
<td>Returns the set of first differences.</td>
</tr>
<tr>
<td>LQ</td>
<td>Returns the LQ factorization of an $n \times m$ matrix.</td>
</tr>
<tr>
<td>LSQ</td>
<td>Returns the minimum norm least squares solution to any system of linear equations.</td>
</tr>
<tr>
<td>LU</td>
<td>Returns the LU decomposition of a square matrix.</td>
</tr>
<tr>
<td>MCALC</td>
<td>Designates a variable as a calculated value (not user-defined).</td>
</tr>
<tr>
<td>MERGE1</td>
<td>Merges the RAM from the card in port 1 with the rest of main user memory.</td>
</tr>
<tr>
<td>Command</td>
<td>Brief Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MINEHUNT</td>
<td>Starts the MINEHUNT game.</td>
</tr>
<tr>
<td>MINIT</td>
<td>Creates the reserved variable $Mpar$.</td>
</tr>
<tr>
<td>MITM</td>
<td>Changes multiple equation menu titles and order.</td>
</tr>
<tr>
<td>MROOT</td>
<td>Solves for one or more variables.</td>
</tr>
<tr>
<td>MSGBOX</td>
<td>Creates a user-defined message box.</td>
</tr>
<tr>
<td>MSOLVR</td>
<td>Gets the Multiple-Equation Solver variable menu for the set of equations defined by $Mpar$.</td>
</tr>
<tr>
<td>MUSER</td>
<td>Designates a variable as user-defined.</td>
</tr>
<tr>
<td>NDIST</td>
<td>Returns the normal probability distribution.</td>
</tr>
<tr>
<td>NOVAL</td>
<td>Place holder for reset and initial values in user-defined dialog boxes.</td>
</tr>
<tr>
<td>NSUB</td>
<td>Provides a way to access the current sublist number during an iteration of a program or command applied using DOSUBS.</td>
</tr>
<tr>
<td>NUMX</td>
<td>Sets the number of x-steps for each y-step in 3D perspective plots.</td>
</tr>
<tr>
<td>NUMY</td>
<td>Sets the number of y-steps across the view volume in 3D perspective plots.</td>
</tr>
<tr>
<td>PARSURFACE</td>
<td>Specifies 3D parameterized surface grip mapping as the plot type.</td>
</tr>
<tr>
<td>PCOEF</td>
<td>Returns the coefficients of a monic polynomial.</td>
</tr>
<tr>
<td>PCOV</td>
<td>Calculates population covariance.</td>
</tr>
<tr>
<td>PCONTOUR</td>
<td>Specifies pseudo-contour as the plot type.</td>
</tr>
<tr>
<td>PEVAL</td>
<td>Evaluates an $n$-degree polynomial at $x$.</td>
</tr>
<tr>
<td>PINIT</td>
<td>Initializes the plug-in card ports.</td>
</tr>
<tr>
<td>PROOT</td>
<td>Returns all roots of an $n$-degree polynomial having real or complex coefficients.</td>
</tr>
</tbody>
</table>
# New Commands Listed Alphabitically (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSDEV</td>
<td>Calculates population standard deviation.</td>
</tr>
<tr>
<td>PVAR</td>
<td>Calculates population variance.</td>
</tr>
<tr>
<td>QR</td>
<td>Returns the QR factorization of an $n \times m$ matrix.</td>
</tr>
<tr>
<td>RANK</td>
<td>Returns the rank of a rectangular matrix.</td>
</tr>
<tr>
<td>RANM</td>
<td>Returns a matrix of random integers.</td>
</tr>
<tr>
<td>RCI</td>
<td>Multiplies a row of a matrix by a constant.</td>
</tr>
<tr>
<td>RCIJ</td>
<td>Multiplies a row of a matrix by a constant, and then adds the product to another row of the matrix.</td>
</tr>
<tr>
<td>RECT</td>
<td>Sets Rectangular coordinate mode.</td>
</tr>
<tr>
<td>REVLIST</td>
<td>Reverses the order of the elements in a list.</td>
</tr>
<tr>
<td>RKF</td>
<td>Computes the solution to an initial value problem for a differential equation, using the Runge-Kutta-Fehlberg method.</td>
</tr>
<tr>
<td>RKFERR</td>
<td>Returns the absolute error estimate for a given step when solving an initial value problem for a differential equation (using RKF method).</td>
</tr>
<tr>
<td>RKFSTEP</td>
<td>Computes the next solution step to an initial value problem for a differential equation.</td>
</tr>
<tr>
<td>ROW+</td>
<td>Inserts an array into a matrix.</td>
</tr>
<tr>
<td>ROW−</td>
<td>Deletes a row from a matrix.</td>
</tr>
<tr>
<td>RREF</td>
<td>Converts a rectangular matrix to reduced row echelon form.</td>
</tr>
<tr>
<td>RRK</td>
<td>Computes the solution to an initial value problem for a differential equation with known partial derivatives.</td>
</tr>
<tr>
<td>RRKSTEP</td>
<td>Computes the next solution step to an initial value problem for a differential equation, and displays the method used to arrive at that result.</td>
</tr>
<tr>
<td>Command</td>
<td>Brief Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RSBERR</td>
<td>Returns an error estimate for a given step when solving an initial values problem for a differential equation (using the Rosenbrock method).</td>
</tr>
<tr>
<td>RSWP</td>
<td>Swaps rows in a matrix.</td>
</tr>
<tr>
<td>SCHUR</td>
<td>Returns the Schur decomposition of a square matrix.</td>
</tr>
<tr>
<td>SEQ</td>
<td>Returns a list of results generated by repeatedly executing an object on a specified range of elements.</td>
</tr>
<tr>
<td>SIDENS</td>
<td>Calculates the intrinsic density of silicon as a function of temperature.</td>
</tr>
<tr>
<td>SLOPEFIELD</td>
<td>Specifies slopefield as the plot type.</td>
</tr>
<tr>
<td>SNRM</td>
<td>Returns the spectral norm of an array.</td>
</tr>
<tr>
<td>SOLVEQN</td>
<td>Starts the solver for a specified set of equations.</td>
</tr>
<tr>
<td>SORT</td>
<td>Sorts the elements in a list in ascending order.</td>
</tr>
<tr>
<td>SPHERE</td>
<td>Sets Spherical coordinate mode.</td>
</tr>
<tr>
<td>SRAD</td>
<td>Returns the spectral radius of a square matrix.</td>
</tr>
<tr>
<td>STREAM</td>
<td>Applies an object to every element in a list.</td>
</tr>
<tr>
<td>SVD</td>
<td>Returns the singular value decomposition of an $n \times m$ matrix.</td>
</tr>
<tr>
<td>SVL</td>
<td>Returns the singular values of an $m \times n$ matrix.</td>
</tr>
<tr>
<td>TAIL</td>
<td>Returns all but the first element of a list or string.</td>
</tr>
<tr>
<td>TDELTA</td>
<td>Calculates a temperature change.</td>
</tr>
<tr>
<td>TEACH</td>
<td>Creates an EXAMPLES subdirectory in the HOME directory and loads HP 48 programming, graphing, and solver examples from ROM into it.</td>
</tr>
<tr>
<td>TINC</td>
<td>Calculates a temperature increment.</td>
</tr>
<tr>
<td>TRACE</td>
<td>Returns the trace of a square matrix.</td>
</tr>
<tr>
<td>Command</td>
<td>Brief Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TVM</td>
<td>Start the TVM solver.</td>
</tr>
<tr>
<td>TVMBEG</td>
<td>Specifies that payments are made at the beginning of compounding periods.</td>
</tr>
<tr>
<td>TVMEND</td>
<td>Specifies that payments are made at the end of compounding periods.</td>
</tr>
<tr>
<td>TVMROOT</td>
<td>Solves for the specified TVM variable using values from the remaining TVM variables.</td>
</tr>
<tr>
<td>VERSION</td>
<td>Returns the software version and copyright message.</td>
</tr>
<tr>
<td>WIREFRAME</td>
<td>Specifies wireframe as the plot type.</td>
</tr>
<tr>
<td>XRECV</td>
<td>Receives an object via XModem.</td>
</tr>
<tr>
<td>XSEND</td>
<td>Sends an object via XModem.</td>
</tr>
<tr>
<td>XVOL</td>
<td>Sets the width of the view volume in the reserved variable $VPAR$.</td>
</tr>
<tr>
<td>XXRNG</td>
<td>Specifies the x range of an input plane (domain) for GRIDMAP and PARSURFACE plots.</td>
</tr>
<tr>
<td>YSLICE</td>
<td>Specifies y-slice cross sections as the plot type.</td>
</tr>
<tr>
<td>YVOL</td>
<td>Sets the depth of the view volume in the reserved variable $VPAR$.</td>
</tr>
<tr>
<td>YYRNG</td>
<td>Specifies the y range of an input plane (domain) for GRIDMAP and PARSURFACE plots.</td>
</tr>
<tr>
<td>ZFACTOR</td>
<td>Calculates the the gas compressibility correction factor for nonideal behavior of a hydrocarbon gas.</td>
</tr>
<tr>
<td>ZVOL</td>
<td>Sets the height of the view volume in the reserved variable $VPAR$.</td>
</tr>
</tbody>
</table>
This appendix contains the following information:

- Object sizes.
- Mathematical simplification rules used by the HP 48.
- Symbolic differentiation patterns used by the HP 48.
- The EquationWriter's expansion rules.
- References used as sources for constants and equations in the HP 48 (other than those in the Equation Library).
The following table lists object types and their size in bytes. (Note that characters in names, strings, and tags use 1 byte each.)

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algebraic</td>
<td>5 + size of included objects</td>
</tr>
<tr>
<td>Backup Object</td>
<td>12 + number of name characters + size of included object</td>
</tr>
<tr>
<td>Binary Integer</td>
<td>13</td>
</tr>
<tr>
<td>Command</td>
<td>2.5</td>
</tr>
<tr>
<td>Complex matrix</td>
<td>$15 + 16 \times \text{number of elements}$</td>
</tr>
<tr>
<td>Complex number</td>
<td>18.5</td>
</tr>
<tr>
<td>Complex vector</td>
<td>$12.5 + 16 \times \text{number of elements}$</td>
</tr>
<tr>
<td>Directory</td>
<td>$6.5 + \text{size of included variables}$</td>
</tr>
<tr>
<td>Graphics Object</td>
<td>$10 + \text{number of rows} \times \text{CEIL(columns/8)}$</td>
</tr>
<tr>
<td>List</td>
<td>5 + size of included objects</td>
</tr>
<tr>
<td>Matrix</td>
<td>$15 + 8 \times \text{number of elements}$</td>
</tr>
<tr>
<td>Program</td>
<td>12.5 + size of included objects</td>
</tr>
<tr>
<td>Quoted global or local name</td>
<td>8.5 + number of characters</td>
</tr>
<tr>
<td>Real number</td>
<td>10.5</td>
</tr>
<tr>
<td>String</td>
<td>5 + number of characters</td>
</tr>
<tr>
<td>Tagged Object</td>
<td>$3.5 + \text{number of tag characters} + \text{size of untagged object}$</td>
</tr>
<tr>
<td>Unit Object</td>
<td>7.5 +</td>
</tr>
<tr>
<td>real magnitude</td>
<td>2.5 or 10.5</td>
</tr>
<tr>
<td>each prefix</td>
<td>6</td>
</tr>
<tr>
<td>each unit name</td>
<td>5 + number of characters</td>
</tr>
<tr>
<td>each $\times,^,/$ or /</td>
<td>2.5</td>
</tr>
<tr>
<td>each exponent</td>
<td>2.5 or 10.5</td>
</tr>
<tr>
<td>Unquoted global or local name</td>
<td>3.5 + number of characters</td>
</tr>
<tr>
<td>Vector</td>
<td>$12.5 + 8 \times \text{number of elements}$</td>
</tr>
<tr>
<td>XLIB name</td>
<td>5.5</td>
</tr>
</tbody>
</table>
## Automatic Simplification Rules

The following tables list the automatic simplification rules for the HP 48.

### Addition and Subtraction

<table>
<thead>
<tr>
<th>Object</th>
<th>Simplified</th>
<th>Object</th>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x-x )</td>
<td>0</td>
<td>( x+(0,0) )</td>
<td>( x )</td>
</tr>
<tr>
<td>( 0+x )</td>
<td>( x )</td>
<td>( x+p )</td>
<td>( x-p )</td>
</tr>
<tr>
<td>((0,0)+x)</td>
<td>( x )</td>
<td>( x-0 )</td>
<td>( x )</td>
</tr>
<tr>
<td>( 0-x )</td>
<td>( \text{NEG}(x) )</td>
<td>( x-(0,0) )</td>
<td>( x )</td>
</tr>
<tr>
<td>((0,0)-x )</td>
<td>( \text{NEG}(x) )</td>
<td>( x+p )</td>
<td>( x-p )</td>
</tr>
<tr>
<td>( x+0 )</td>
<td>( x )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Multiplication and Division

<table>
<thead>
<tr>
<th>Object</th>
<th>Simplified</th>
<th>Object</th>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{INV}(i) )</td>
<td>( -i )</td>
<td>( x \times (1,0) )</td>
<td>( x )</td>
</tr>
<tr>
<td>( y \times \text{INV}(x) )</td>
<td>( y/x )</td>
<td>( x \times (-1) )</td>
<td>( \text{NEG}(x) )</td>
</tr>
<tr>
<td>( y/\text{INV}(x) )</td>
<td>( y \times x )</td>
<td>( x \times (-1,0) )</td>
<td>( \text{NEG}(x) )</td>
</tr>
<tr>
<td>( 0 \times x )</td>
<td>0</td>
<td>( x/1 )</td>
<td>( x )</td>
</tr>
<tr>
<td>((0,0) \times x)</td>
<td>( (0,0) )</td>
<td>( x/(1,0) )</td>
<td>( x )</td>
</tr>
<tr>
<td>( i \times i )</td>
<td>( -1 )</td>
<td>( x/(-1) )</td>
<td>( \text{NEG}(x) )</td>
</tr>
<tr>
<td>( 1 \times x )</td>
<td>( x )</td>
<td>( x/(-1,0) )</td>
<td>( \text{NEG}(x) )</td>
</tr>
<tr>
<td>((1,0) \times x )</td>
<td>( x )</td>
<td>( 0/x )</td>
<td>0</td>
</tr>
<tr>
<td>((-1) \times x)</td>
<td>( \text{NEG}(x) )</td>
<td>( (0,0)/x )</td>
<td>( (0,0) )</td>
</tr>
<tr>
<td>((-1,0) \times x )</td>
<td>( \text{NEG}(x) )</td>
<td>( 1/x )</td>
<td>( \text{INV}(x) )</td>
</tr>
<tr>
<td>( x \times 0 )</td>
<td>0</td>
<td>( (1,0) x )</td>
<td>( \text{INV}(x) )</td>
</tr>
<tr>
<td>( x \times (0,0) )</td>
<td>( (0,0) )</td>
<td>((-1)/x )</td>
<td>( -\text{INV}(x) )</td>
</tr>
<tr>
<td>( x \times 1 )</td>
<td>( x )</td>
<td>((-1,0)/x )</td>
<td>( -\text{INV}(x) )</td>
</tr>
<tr>
<td>Object</td>
<td>Simplified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$1^x$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(1,0)^x$</td>
<td>$(1,0)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{SQ}(\sqrt{x})$</td>
<td>$x$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{SQ}(y^x)$</td>
<td>$y^{2 \times x}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{SQ}(i)$</td>
<td>$-1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x^0$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x^{(0,0)}$</td>
<td>$(1,0)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x^1$</td>
<td>$x$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^{(1,0)}$</td>
<td>$x$</td>
</tr>
<tr>
<td>$x^{(-1)}$</td>
<td>$\text{INV}(x)$</td>
</tr>
<tr>
<td>$x^{(-1,0)}$</td>
<td>$\text{INV}(x)$</td>
</tr>
<tr>
<td>$(\sqrt{x})^2$</td>
<td>$x$</td>
</tr>
<tr>
<td>$(\sqrt{x})^{(2,0)}$</td>
<td>$x$</td>
</tr>
<tr>
<td>$i^2$</td>
<td>$-1$</td>
</tr>
<tr>
<td>$i^{(2,0)}$</td>
<td>$(-1,0)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ABS}(\text{ABS}(x))$</td>
<td>$\text{ABS}(x)$</td>
</tr>
<tr>
<td>$\text{ABS}(\text{NEG}(x))$</td>
<td>$\text{ABS}(x)$</td>
</tr>
<tr>
<td>$\text{CONJ}(\text{CONJ}(x))$</td>
<td>$x$</td>
</tr>
<tr>
<td>$\text{CONJ}(\text{IM}(x))$</td>
<td>$\text{IM}(x)$</td>
</tr>
<tr>
<td>$\text{CONJ}(\text{RE}(x))$</td>
<td>$\text{RE}(x)$</td>
</tr>
<tr>
<td>$\text{CONJ}(i)$</td>
<td>$-i$</td>
</tr>
<tr>
<td>$\text{IM}(\text{CONJ}(x))$</td>
<td>$-\text{IM}(x)$</td>
</tr>
<tr>
<td>$\text{IM}(\text{IM}(x))$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{IM}(\text{RE}(x))$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{IM}(p)$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{IM}(i)$</td>
<td>1</td>
</tr>
<tr>
<td>$\text{MAX}(x,x)$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{MIN}(x,x)$</td>
<td>$x$</td>
</tr>
<tr>
<td>$\text{MOD}(0,x)$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{MOD}(x,x)$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{MOD}(x,0)$</td>
<td>$x$</td>
</tr>
<tr>
<td>$x \mod y$</td>
<td>$x \mod y$</td>
</tr>
<tr>
<td>$\text{RE}(\text{CONJ}(x))$</td>
<td>$\text{RE}(x)$</td>
</tr>
<tr>
<td>$\text{RE}(\text{IM}(x))$</td>
<td>$\text{IM}(x)$</td>
</tr>
<tr>
<td>$\text{RE}(\text{RE}(x))$</td>
<td>$\text{RE}(x)$</td>
</tr>
<tr>
<td>$\text{RE}(\pi)$</td>
<td>$\pi$</td>
</tr>
<tr>
<td>$\text{RE}(i)$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{SIGN}(\text{SIGN}(x))$</td>
<td>$\text{SIGN}(x)$</td>
</tr>
</tbody>
</table>
Symbolic Integration Patterns

This table lists the symbolic integration patterns used by the HP 48. These are the integrands that the HP 48 can integrate symbolically.

\( \phi \) is a linear function of the variable of integration. The antiderivatives should be divided by the first-order coefficient in \( \phi \) to reduce the expression to its simplest form. Also, patterns beginning with 1/ match INV: for example, 1/\( \phi \) is the same as INV(\( \phi \)).

### Symbolic Integration

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Antiderivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS(( \phi ))</td>
<td>( \phi \times \text{ACOS}(\phi) - \sqrt{(1-\phi^2)} )</td>
</tr>
<tr>
<td>ALOG(( \phi ))</td>
<td>.434294481904 \times \text{ ALOG}(\phi)</td>
</tr>
<tr>
<td>ASIN(( \phi ))</td>
<td>( \phi \times \text{ASIN}(\phi) + \sqrt{(1-\phi^2)} )</td>
</tr>
<tr>
<td>ATAN(( \phi ))</td>
<td>( \phi \times \text{ATAN}(\phi) - \ln(1+\phi^2)/2 )</td>
</tr>
<tr>
<td>COS(( \phi ))</td>
<td>\text{SIN}(\phi)</td>
</tr>
<tr>
<td>1/(COS(( \phi ))xSIN(( \phi )))</td>
<td>\text{LN}(\text{TAN}(\phi))</td>
</tr>
<tr>
<td>COSH(( \phi ))</td>
<td>\text{SINH}(\phi)</td>
</tr>
<tr>
<td>1/(COSH(( \phi ))xSINH(( \phi )))</td>
<td>\text{LN}(\text{TANH}(\phi))</td>
</tr>
<tr>
<td>1/(COSH(( \phi ))^2)</td>
<td>\text{TANH}(\phi)</td>
</tr>
<tr>
<td>EXP(( \phi ))</td>
<td>\text{EXP}(\phi)</td>
</tr>
<tr>
<td>EXPM(( \phi ))</td>
<td>\text{EXP}(\phi) - \phi</td>
</tr>
<tr>
<td>LN(( \phi ))</td>
<td>( \phi \times \text{LN}(\phi) - \phi )</td>
</tr>
<tr>
<td>LOG(( \phi ))</td>
<td>.434294481904 \times \phi \times \text{LN}(\phi) - \phi</td>
</tr>
<tr>
<td>SIGN(( \phi ))</td>
<td>\text{ABS}(\phi)</td>
</tr>
<tr>
<td>SIN(( \phi ))</td>
<td>-\text{COS}(\phi)</td>
</tr>
<tr>
<td>1/(SIN(( \phi ))xCOS(( \phi )))</td>
<td>\text{LN}(\text{TAN}(\phi))</td>
</tr>
<tr>
<td>1/(SIN(( \phi ))xTAN(( \phi )))</td>
<td>-\text{INV}(\text{SIN}(\phi))</td>
</tr>
<tr>
<td>1/(SIN(( \phi ))xTAN(( \phi )))</td>
<td>-\text{INV}(\text{SIN}(\phi))</td>
</tr>
<tr>
<td>1/(SIN(( \phi ))^2)</td>
<td>-\text{INV}(\text{TAN}(\phi))</td>
</tr>
<tr>
<td>SINH(( \phi ))</td>
<td>\text{COSH}(\phi)</td>
</tr>
<tr>
<td>1/(SINH(( \phi ))^2)</td>
<td>-\text{INV}(\text{SIN}(\phi))</td>
</tr>
</tbody>
</table>
### Symbolic Integration (continued)

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Antiderivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1/(\text{SINH}(\phi) \times \text{COSH}(\phi))$</td>
<td>$\ln(\text{TANH}(\phi))$</td>
</tr>
<tr>
<td>$1/(\text{SINH}(\phi) \times \text{TANH}(\phi))$</td>
<td>$-\ln(\text{TANH}(\phi))$</td>
</tr>
<tr>
<td>$\text{SQ}(\phi)$</td>
<td>$\phi^3/3$</td>
</tr>
<tr>
<td>$\text{TAN}(\phi)^2$</td>
<td>$\text{TAN}(\phi) - \phi$</td>
</tr>
<tr>
<td>$\text{TAN}(\phi)$</td>
<td>$-\ln(\cos(\phi))$</td>
</tr>
<tr>
<td>$\text{TAN}(\phi)/\cos(\phi)$</td>
<td>$\ln(\cos(\phi))$</td>
</tr>
<tr>
<td>$1/\text{TAN}(\phi)$</td>
<td>$\ln(\sin(\phi))$</td>
</tr>
<tr>
<td>$1/\text{TAN}(\phi) \times \sin(\phi)$</td>
<td>$-\ln(\sin(\phi))$</td>
</tr>
<tr>
<td>$\text{TANH}(\phi)$</td>
<td>$\ln(\cosh(\phi))$</td>
</tr>
<tr>
<td>$\text{TANH}(\phi)/\cosh(\phi)$</td>
<td>$\ln(\cosh(\phi))$</td>
</tr>
<tr>
<td>$1/\text{TANH}(\phi)$</td>
<td>$\ln(\sinh(\phi))$</td>
</tr>
<tr>
<td>$1/\text{TANH}(\phi) \times \sinh(\phi)$</td>
<td>$-\ln(\sinh(\phi))$</td>
</tr>
<tr>
<td>$\sqrt{\phi}$</td>
<td>$2 \phi^{1.5}/3$</td>
</tr>
<tr>
<td>$1/\sqrt{\phi}$</td>
<td>$2 \sqrt{\phi}$</td>
</tr>
<tr>
<td>$1/(2 \sqrt{\phi})$</td>
<td>$2 \sqrt{\phi} \times .5$</td>
</tr>
<tr>
<td>$\phi^z$ (z symbolic)</td>
<td>IFTE(z==−1, $\ln(\phi), \phi^{(z+1)}/(z+1)$)</td>
</tr>
<tr>
<td>$\phi^z$ (z real, $\neq 0,-1$)</td>
<td>$\phi^{(z+1)}/(z+1)$</td>
</tr>
<tr>
<td>$\phi^{-1}$</td>
<td>$\ln(\phi)$</td>
</tr>
<tr>
<td>$1/\phi$</td>
<td>$\ln(\phi)$</td>
</tr>
<tr>
<td>$1/(1-\phi^2)$</td>
<td>$\text{ATANH}(\phi)$</td>
</tr>
<tr>
<td>$1/(1+\phi^2)$</td>
<td>$\text{ATAN}(\phi)$</td>
</tr>
<tr>
<td>$1/(\phi^2+1)$</td>
<td>$\text{ATAN}(\phi)$</td>
</tr>
<tr>
<td>$1/((\sqrt{\phi-1}) \times \sqrt{\phi+1}))$</td>
<td>$\text{ACOSH}(\phi)$</td>
</tr>
<tr>
<td>$1/\sqrt{1-\phi^2)}$</td>
<td>$\text{ASIN}(\phi)$</td>
</tr>
<tr>
<td>$1/\sqrt{1+\phi^2)}$</td>
<td>$\text{ASINH}(\phi)$</td>
</tr>
<tr>
<td>$1/\sqrt{\phi^2+1)}$</td>
<td>$\text{ASINH}(\phi)$</td>
</tr>
</tbody>
</table>
## Trigonometric Expansions

The following tables list expansions for trigonometric functions in Radians mode when using the $\rightarrow$DEF, TRG*, and $\rightarrow$TRG operations. These operations appear in the EquationWriter RULES menu.

### $\rightarrow$DEF Expansions

<table>
<thead>
<tr>
<th>Function</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sin(x)$</td>
<td>$\frac{\exp(x \times i) - \exp(-(x \times i))}{2 \times i}$</td>
</tr>
<tr>
<td>$\cos(x)$</td>
<td>$\frac{\exp(x \times i) + \exp(-(x \times i))}{2}$</td>
</tr>
<tr>
<td>$\tan(x)$</td>
<td>$\frac{\exp(x \times i \times 2) - 1}{(\exp(x \times i \times 2) + 1) \times i}$</td>
</tr>
<tr>
<td>$\sinh(x)$</td>
<td>$-(\sin(x \times i) \times i)$</td>
</tr>
<tr>
<td>$\cosh(x)$</td>
<td>$\cos(x \times i)$</td>
</tr>
<tr>
<td>$\tanh(x)$</td>
<td>$\tan(x \times i) \times -i$</td>
</tr>
<tr>
<td>$\arcsin(x)$</td>
<td>$-i \times \ln(\sqrt{1 - x^2} + i \times x)$</td>
</tr>
<tr>
<td>$\arccos(x)$</td>
<td>$\frac{\pi}{2} + i \times \ln(\sqrt{1 - x^2} + i \times x)$</td>
</tr>
<tr>
<td>$\arctan(x)$</td>
<td>$-i \times \ln\left(\frac{1 + i \times x}{\sqrt{1 + x^2}}\right)$</td>
</tr>
<tr>
<td>$\arcsinh(x)$</td>
<td>$-\ln(\sqrt{1 + x^2} - x)$</td>
</tr>
<tr>
<td>$\arccosh(x)$</td>
<td>$\sqrt{-\left(\frac{\pi}{2} + i \times \ln(\sqrt{1 - x^2} + i \times x)\right)^2}$</td>
</tr>
<tr>
<td>$\arctanh(x)$</td>
<td>$-\ln\left(\frac{1 - x}{\sqrt{1 - x^2}}\right)$</td>
</tr>
</tbody>
</table>
### TRG⁺ Expansions

<table>
<thead>
<tr>
<th>Function</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sin(x + y)$</td>
<td>$\sin(x) \times \cos(y) + \cos(x) \times \sin(x)$</td>
</tr>
<tr>
<td>$\cos(x + y)$</td>
<td>$\cos(x) \times \cos(y) + \sin(x) \times \sin(y)$</td>
</tr>
<tr>
<td>$\tan(x + y)$</td>
<td>$\frac{\tan(x) + \tan(y)}{1 - \tan(x) \times \tan(y)}$</td>
</tr>
<tr>
<td>$\sinh(x + y)$</td>
<td>$\sinh(x) \times \cosh(y) + \cosh(x) \times \sinh(x)$</td>
</tr>
<tr>
<td>$\cosh(x + y)$</td>
<td>$\cosh(x) \times \cosh(y) + \sinh(x) \times \sinh(y)$</td>
</tr>
<tr>
<td>$\tanh(x + y)$</td>
<td>$\frac{\tanh(x) + \tanh(y)}{1 + \tanh(x) \times \tanh(y)}$</td>
</tr>
</tbody>
</table>

### →TRG Expansion

<table>
<thead>
<tr>
<th>Function</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\exp(x)$</td>
<td>$\cos\left(\frac{x}{i}\right) + \sin\left(\frac{x}{i}\right) \times i$</td>
</tr>
</tbody>
</table>
Source References

The following references were used as sources for many of the constants and equations used in the HP 48. (See “References” in chapter 4, “Equation Reference,” for the references used as sources for the Equation Library.)


Parallel Processing with Lists

Parallel processing is the idea that, generally, if a command can be applied to one or more individual arguments, then it can also be extended to be applied to one or more sets of arguments.

Some examples:

- 5 INV returns 2, so \( \{ 4 \ 5 \ 8 \} \) INV returns \( \{ .25 \ .2 \ .125 \} \).
- 4 5 \* returns 20, so \( \{ 4 \ 5 \ 6 \} \) \( \{ 5 \ 6 \ 7 \} \) \* returns \( \{ 20 \ 30 \ 42 \} \), and \( \{ 4 \ 5 \ 6 \} \) 5 \* returns \( \{ 20 \ 25 \ 30 \} \).

General Rules for Parallel Processing

As a rule-of-thumb, a given command can use parallel list processing if all the following are true:

- The command checks for valid argument types. Commands that apply to all object types, such as DUP, SWAP, ROT, and so forth, do not use parallel list processing.
- The command takes exactly one, two, three, four, or five arguments, none of which may itself be a list. Commands that use an indefinite number of arguments (such as →LIST) do not use parallel list processing.
- The command isn’t a programming branch command (IF, FOR, CASE, NEXT, and so forth).

The remainder of this appendix describes how the many and various commands available on the HP 48 are grouped with respect to parallel processing.

Group 1: Commands that cannot parallel process

A command must take arguments before it can parallel process, since a zero-argument command (such as RAND, VARS, or REC) has no arguments with which to form a group.
Group 2: Commands that must use DOLIST to parallel process

This group of commands cannot use parallel processing directly, but can be "coerced" into it using the DOLIST command (see "Using DOLIST for Parallel Processing" later in this appendix). This group consists of several subgroups:

- **Stack manipulation commands.** A stack manipulation command cannot parallel process because the stack is manipulated as a whole and list objects are treated the same as any other object. Stack commands (such as DROPN) that take level 1 arguments will not accept level 1 list arguments.

- **Commands that operate on lists as wholes.** Certain commands accept lists as arguments but treat them no differently than any other data object. They perform their function on the object as a whole without respect to its elements. For example, →STR converts the entire list object to a string rather than converting each individual element, and the == command tests the level 1 object against the level 2 object regardless of the objects' types.

- **List manipulation commands.** List manipulation commands will not parallel process since they operate on list arguments as lists rather than as sets of parallel data. However, a list manipulation command can be forced to parallel process lists of lists by using the the DOLIST command. For example, \{ \{ 1 2 3 \} \{ 4 5 6 \} \} TLIST » DOLIST returns \{ 6 1 2 0 3 \}.

- **Other commands that have list arguments.** Because a list can hold any number of objects of any type, it is commonly used to hold a variable number of parameters of various types. Some commands accept such lists, and because of this are insensitive to parallel processing, except by using DOLIST.

- **Index-oriented commands.** Many array commands either establish the size of an array in rows and columns or manipulate individual elements by their row and column indices. These commands expect these row and column indices to be real number pairs collected in lists. For example, \{ 3 4 \} RANM will generate a random integer matrix having 3 rows and 4 columns. Since these commands can normally use lists as arguments, they cannot perform parallel processing, except by using DOLIST.

- **Program control commands.** Program control structures and commands do not perform parallel processing and cannot be forced
to do so. However, programs containing these structures can be made to parallel process by using DOLIST. For example, \(\{1 \ 2 \ 3 \ 4 \ 5 \ 6 \} \ « \text{IF} \ \text{DUP} \ 3 \ \text{≤} \ \text{THEN} \ \text{DROP} \ \text{END} » \) DOLIST returns \(\{3 \ 4 \ 5 \ 6 \}\).

**Group 3: Commands that sometimes work with parallel processing**

Graphics commands that can take pixel coordinates as arguments expect those coordinates to be presented as two-element lists of binary integers. Since these commands can normally use lists as arguments, they cannot parallel process, except by using DOLIST.

For the two-argument graphics commands (BOX, LINE, TLINE), if either argument is not a list (a complex number, for example), then the commands will parallel process, taking the list argument to be multiple complex number coordinates. For example, \((0,0) \ (1,1) \ (3,2) » \) LINE will draw two lines—between (0,0) and (1,1) and between (0,0) and (3,2).

**Group 4: ADD and +**

On HP 48S and HP 48SX calculators, the + command has been used to append lists or to append elements to lists. Thus \(\{1 \ 2 \ 3 \} + 4 \) returns \(\{1 \ 2 \ 3 \ 4 \}\). With the advent of parallel processing in the HP 48G series, the ADD command was created to perform parallel addition instead of +.

This has several ramifications:

- To add two lists in parallel, you must do one of the following:
  - Use ADD from the \(\textbf{MTH} \ \textbf{LIST}\) menu.
  - Create a custom menu containing the ADD command.
  - Assign the ADD command to a user-defined key.

- User programs must be written using ADD instead of + if the program is to be able to perform direct parallel processing, or written with + and applied to their arguments by using DOLIST. For example, programs such as \(\{ \text{«} \ \text{×} \ « \text{ADD} \ 2 » \ \text{+} \ \text{×} + 2 \} \) will produce list concatenation when \(x\) is a list rather than parallel addition, unless rewritten as \(\{ \text{«} \ \text{×} \ « \text{ADD} \ 2 » \ \text{+} \ \text{×} + 2 \} \).

- Algebraic expressions capable of calculating with variables containing lists (including those intended to become user-defined functions) cannot be created in RPN syntax since using...
ADD to add two symbolic arguments concatenates the arguments with + rather than with ADD. For example, '\(X\) DUP 2 ^ SWAP 4 * ADD 'F(X)' SWAP = produces 'F(X)=x^2+4*x' rather than 'F(X)=x^2 ADD 4*x'.

**Group 5: Commands that set modes/states**

Commands that store values in system-specific locations so as to control certain modes and machine states can generally be used to parallel process data. The problem is that each successive parameter in the list cancels the setting established by the previous parameter. For example, \(\{1 \ 2 \ 3 \ 4 \ 5\} \text{ FIX}\) is effectively the same as \(5 \text{ FIX}\).

**Group 6: One-argument, one-result commands**

These commands are the easiest to use with parallel processing. Simply provide the command with a list of arguments instead of the expected single argument. Some examples:

\[
\begin{align*}
\{1 \ -2 \ 3 \ -4\} \text{ ABS} & \text{ returns } \{1 \ 2 \ 3 \ 4\} \\
\text{DEG} \{0 \ 30 \ 60 \ 90\} \text{ SIN} & \text{ returns } \{0 \ .5 \ .866025403784 \ 1\} \\
\{1 \ \text{A} \ '\text{SIN(Z)}'\} \text{ INV} & \text{ returns } \{1 \ '\text{INV(A)}' \ '\text{INV(SIN(Z))}'\}
\end{align*}
\]

**Group 7: Two-argument, one-result commands**

Two-argument commands can operate in parallel in any of three different ways:

- \(\{\text{list}\} \times \{\text{list}\}\)
- \(\{\text{list}\} \times \text{object}\)
- \(\text{object} \times \{\text{list}\}\)

In the first form, parallel elements are combined by the command:

\[
\{1 \ 2 \ 3\} \times \{4 \ 5 \ 6\} \times \text{returns } \{.04 \ .1 \ .18\}.
\]

In the second form, the level 1 object is combined with each element in the level 2 list in succession:

\[
\{1 \ 2 \ 3\} \times \text{30 CH} \text{ returns } \{2900 \ 1400 \ 900\}.
\]

In the third form, the level 2 object is combined with each element of the level 1 list in succession:

\[
50 \times \{1 \ 2 \ 3\} \times \text{T} \text{ returns } \{2 \ 4 \ 6\}.
\]
Group 8: Multiple-argument, one-result commands

Commands that take multiple (3, 4, or 5) arguments can perform parallel processing only if all arguments are lists. For example, \( \text{ROOT} \{ \text{'SIN(x)'} , \text{'COS(x)'} , \text{'TAN(x)'} \} \{ \text{x xx xx} \} \{ \text{0 0 0} \} \) returns \( \{ 0 90 0 \} \). Notice that lists must be used even though the level 1 and level 2 lists each contain multiples of the same element.

Group 9: Multiple-result commands

Any command that allows parallel processing, but produces multiple results from its input data, will return its results as a single list. For example, \( \{1 2 3\} \{4 5 6\} \) \( \text{C-R} \) produces \( \{1 4 2 5 3 6\} \) rather than the more expected \( \{1 2 3\} \{4 5 6\} \).

The following UNMIX program will unmix the data given the number of expected result lists:

\[
\begin{align*}
\text{\texttt{\textasciitilde OVER SIZE + 1 n s}} \\
\text{\texttt{\textasciitilde 1 n}} \\
\text{\texttt{FOR j j s}} \\
\text{\texttt{FOR i 1 i GET n}} \\
\text{\texttt{STEP s n \rightarrow LIST}} \\
\text{\texttt{NEXT}} \\
\text{\texttt{\textasciitilde \}}} \\
\text{\texttt{\textasciitilde \}}}
\end{align*}
\]

Taking \( \{1 4 2 5 3 6\} \) from above as the result of \( \text{C-R} \) (a command which should return two results), \( \text{2 UNMIX} \) gives \( \{1 2 3\} \{4 5 6\} \).

Group 10: Quirky commands

A few commands behave uniquely with respect to parallel processing:

- **DELALARM.** This command can take a list of arguments. Note, however, that deletions from early in the alarm list will change the alarm indices of the later alarm entries. Thus, if there are only three alarms, \( \{1 3\} \) \text{DELALARM} will cause an error, whereas \( \{3 1\} \) \text{DELALARM} will not.

- **DOERR.** This command forces an error state that causes all running programs and commands to halt. Thus, even though providing the command with a list argument will cause the command to perform parallel processing, the first error state
will cause the command to abort and none of the rest of the list arguments will be used.

- **FREE, MERGE.** Only port 1 can be freed or merged on the HP 48GX. Thus, even though a list argument is acceptable, an error will occur for any list except `{ 1 }`.

- **RESTORE.** This command performs a system warmstart after installing the backup object into memory. All functions are terminated at that time. Thus, only the first backup object in a list will be restored.

- ** _(Attach Unit)._** This command will create unit objects in parallel only if level 1 contains a list. Thus `{ 1 { f t 1 i n m } _ _ }` produces `{ 1 _ f t 1 _ i n 1 _ m }` while `{ 1 2 3 } 'm' _ _ produces an error.

- **STO+.** STO+ performs parallel list addition only if both arguments are lists. If one argument is a list and the other is not, STO+ appends the non-list argument to each element in the list.

- **STO-, STO*, STO/.** These commands perform parallel processing if both arguments are lists, but fail otherwise.

### Using DOLIST for Parallel Processing

Almost any command or user program can be made to work in parallel over a list or lists of data by using the DOLIST command. Use DOLIST as follows.

- Level 1 must contain a command, a program object, or the name of a variable that contains a command or program object.

- Level 2 must contain an argument count unless the level 1 object is a command that accepts parallel processing, a program containing only one command that accepts parallel processing, or a user-defined function. In these special cases, Level 2 contains the first of the list arguments.

- If level 2 was the argument count, then level 3 is the first of the argument lists. Otherwise, levels 2 through \( n \) are the argument lists.
As an example, the following program takes three objects from the stack, tags them with the names \( a \), \( b \), and \( c \), and displays them one after the other in line 1 of the display.

```plaintext
« \to \ a \ b \ c
\{ \ a \ b \ c \} \ DUP \ « \ EVAL \ » \ DOLIST
\SWAP \ « \ \to \TAG \ » \ DOLIST
\CRLCD \ 1 \ « \ 1 \ DISP \ 1 \ WAIT \ » \ DOLIST
»
»
```
Index

Special characters

® character, 1-10

A

absolute value, 3-5
alarms
  acknowledging, 3-6
  deleting, 3-78
  finding, 3-116
  index number, 3-116
  recalling, 3-262
  storing, 3-326
ALG annunciator, 1-9
algebraic
  linear structure, 1-20
Algebraic/Program-entry mode, 1-9, 1-63
algebraics
  action in programs, 1-2
  collecting terms, 3-56
  comparing, 1-19
  conditional testing, 1-22
  editing in programs, 1-9
  expanding, 3-109
  in local variable structure, 1-3, 1-11
  isolating variables, 3-158
  rearranging, 3-104, 3-158
  rearranging programmatically, 2-19
  simplifying, 3-56
  testing for linearity, 3-168
  tests in, 1-19
algebraic syntax
  conditional testing, 1-22
  in local variable structures, 1-4
  test commands, 1-17, 1-19
alpha keyboard
  automatically locking, 1-63
amortization (TVM), 3-12
angle mode
  setting, 3-78, 3-131, 3-256
angles
  converting units, 3-99, 3-294
angular mechanics, 4-29
angular motion, 4-48
animation, 2-45, 2-56, 3-14
annunciators
  user flags, 1-42
applications, 1-79
archives
  creating, 3-18
arcs, 3-17
arguments
  comparing, 3-183, 3-191
  recalling last, 3-162, 3-163
  verifying, 2-36
arrays
  applying a program to, 2-29
  column norm, 3-53
  complex conjugates, 3-62
  constant, 3-59
<table>
<thead>
<tr>
<th>Term</th>
<th>Page Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>creating from stack</td>
<td>3-20, 3-369</td>
</tr>
<tr>
<td>deleting columns</td>
<td>3-55</td>
</tr>
<tr>
<td>disassembling</td>
<td>3-19</td>
</tr>
<tr>
<td>disassembling complex</td>
<td>3-71</td>
</tr>
<tr>
<td>extracting elements</td>
<td>3-127, 3-129</td>
</tr>
<tr>
<td>Fourier transforms</td>
<td>3-115</td>
</tr>
<tr>
<td>inserting columns</td>
<td>3-54</td>
</tr>
<tr>
<td>inserting diagonals</td>
<td>3-84</td>
</tr>
<tr>
<td>inverse Fourier transform</td>
<td>3-147</td>
</tr>
<tr>
<td>manipulating</td>
<td>2-16, 2-49</td>
</tr>
<tr>
<td>maximum and minimum elements</td>
<td>2-22</td>
</tr>
<tr>
<td>redimensioning</td>
<td>3-266</td>
</tr>
<tr>
<td>replacing elements in</td>
<td>3-244, 3-246</td>
</tr>
<tr>
<td>residual</td>
<td>3-291</td>
</tr>
<tr>
<td>row norm</td>
<td>3-280</td>
</tr>
<tr>
<td>sorting elements</td>
<td>2-23</td>
</tr>
<tr>
<td>spectral norm of</td>
<td>3-314</td>
</tr>
<tr>
<td>swapping columns</td>
<td>3-70</td>
</tr>
<tr>
<td>symbolic</td>
<td>2-29</td>
</tr>
<tr>
<td>axes (plots)</td>
<td></td>
</tr>
<tr>
<td>controlling</td>
<td>3-33</td>
</tr>
<tr>
<td>including</td>
<td>3-95</td>
</tr>
<tr>
<td>labeling</td>
<td>3-162</td>
</tr>
<tr>
<td>beeper</td>
<td></td>
</tr>
<tr>
<td>in programs</td>
<td>1-71</td>
</tr>
<tr>
<td>sounding</td>
<td>3-37</td>
</tr>
<tr>
<td>specifying tone and duration</td>
<td>3-37</td>
</tr>
<tr>
<td>Bernoulli equation</td>
<td>4-23</td>
</tr>
<tr>
<td>Bessel functions</td>
<td>2-43</td>
</tr>
<tr>
<td>binary integers</td>
<td></td>
</tr>
<tr>
<td>comparing</td>
<td>1-19</td>
</tr>
<tr>
<td>converting to floating-point</td>
<td>3-42</td>
</tr>
<tr>
<td>custom display</td>
<td>2-7</td>
</tr>
<tr>
<td>representing flags</td>
<td>1-44</td>
</tr>
<tr>
<td>shifting one bit right</td>
<td>3-25</td>
</tr>
<tr>
<td>wordsize</td>
<td>1-19, 3-335</td>
</tr>
<tr>
<td>binary wordsize</td>
<td></td>
</tr>
<tr>
<td>recalling</td>
<td>3-265</td>
</tr>
<tr>
<td>bipolar transistors</td>
<td>4-73</td>
</tr>
<tr>
<td>bit</td>
<td></td>
</tr>
<tr>
<td>rotate left</td>
<td>3-278</td>
</tr>
<tr>
<td>rotate right</td>
<td>3-285</td>
</tr>
<tr>
<td>shift left</td>
<td>3-310</td>
</tr>
<tr>
<td>shift right</td>
<td>3-317</td>
</tr>
<tr>
<td>black-body emissive power</td>
<td>3-112</td>
</tr>
<tr>
<td>black body radiation</td>
<td>4-42</td>
</tr>
<tr>
<td>boxes</td>
<td>3-40</td>
</tr>
<tr>
<td>branch cuts</td>
<td>3-7, 3-9, 3-21, 3-23, 3-26, 3-29</td>
</tr>
<tr>
<td>branching structures</td>
<td></td>
</tr>
<tr>
<td>conditional structures</td>
<td>1-20, 1-53, 3-101</td>
</tr>
<tr>
<td>ending</td>
<td>3-102</td>
</tr>
<tr>
<td>loop structures</td>
<td>1-27</td>
</tr>
<tr>
<td>program element</td>
<td>1-3</td>
</tr>
<tr>
<td>BRCH menu</td>
<td>1-20, 1-27</td>
</tr>
<tr>
<td>Brewster angle</td>
<td>4-27</td>
</tr>
<tr>
<td>buckling</td>
<td>4-3</td>
</tr>
<tr>
<td>buffer (serial)</td>
<td></td>
</tr>
<tr>
<td>clearing</td>
<td>3-51</td>
</tr>
<tr>
<td>sizing data in</td>
<td>3-40</td>
</tr>
</tbody>
</table>

()`Index-2`
byte
  rotate left, 3-278
  rotate right, 3-285
  shift left, 3-311
  shift right, 3-318

C
  calculator
    turning off, 1-82
  calculator clock, 2-5
  cantilevers, 4-1
  capacitor, 4-16, 4-17, 4-19, 4-20
  “case” branching, 1-22, 2-38, 2-49, 3-42
  case structures, 2-49
  centripetal force, 4-29
  character codes
    remapping to match HP 82240A, 3-211
    to characters, 3-48
  characters
    codes, 3-204
    character translation, 3-352
    checksums, 3-41
    specifying type of, 3-49
    verify programs, 2-1
  chi-square distribution, 3-363
  choose boxes
    custom, 1-68, 3-46
    in programs, 1-68
  circle, 4-59, 4-79
  circular motion, 4-49
  clearing
    command line, 3-50
    directories, 3-53
    display, 1-74
    flags, 1-42, 3-45
    stack, 3-50
    stack display, 3-51
    subdirectories, 3-52
    variables, 3-52, 3-53
  clock
    adjusting, 3-50
  coefficients
    of monic polynomials, 3-220
    regression, 3-177
  collisions, 4-30
  column operations
    converting matrices into columns, 3-54
    converting vectors into elements, 3-54
    creating matrices from columns, 3-56
    creating vectors from elements, 3-56
    deleting, 3-55
    inserting, 3-54
  columns, 4-1
  combinations, 3-58
  command line
    clearing, 3-50
    during program input, 1-62
  commands
    applying to list elements, 3-92, 3-102, 3-203
    applying to lists, 3-91
    in programs, 1-2
  comments, 1-10
  comparing objects, 3-294
  comparison functions, 1-17, 1-19
  compiled local variable structures
    defining procedure, 1-15
  complex numbers
    conjugates, 3-62
    disassembling, 3-71
    imaginary parts, 3-150
    polar angle $\theta$, 3-19
    real parts, 3-267
  computer
    creating programs on, 1-10
conditional commands, 1-20, 1-21, 1-22
conditionals
   nested, 2-23, 2-26, 2-36
conditional structures
   "case" branching, 1-22, 3-42
   conditional commands, 1-20
   ending, 3-102
   error branching, 1-53, 1-54, 3-146
   examples, 1-23
   "if" branching, 1-20, 1-21, 1-22, 1-53, 1-54, 3-101, 3-144, 3-148, 3-149
   program element, 1-3
   test commands in, 1-17, 1-20
conduction, 4-39, 4-41
cone, 4-64
conjugates, 3-62
   of matrices, 3-353
   of objects, 3-300
constants
   symbolic, 3-99, 3-143, 3-183, 3-193
Constants Library
   opening, 3-63
continuing execution, 1-47, 1-48, 1-57, 1-59, 3-64
convection, 4-40, 4-41
converting base units, 3-360
coordinate modes
   specifying, 3-70, 3-269, 3-316
coordinates
   pixels to user units, 3-250
   specifying for PICT, 3-231
   user units to pixels, 3-71
coordinates of PICT
   specifying, 3-231
correlation (statistical), 3-65
Coulomb's law, 4-11

counters
   loop structures, 1-29, 1-31, 1-33, 1-35
   negative steps, 1-31, 1-35, 1-39
   stepping, 1-39
covariance, 3-222
creating 2D vectors, 3-369
creating 3D vectors, 3-370
critical angle, 4-51
current, 4-9, 4-43, 4-67
cursor (command line), 1-63
curve fitting, 3-37, 3-110, 3-166, 3-167, 3-176, 3-250
custom menus
   creating, 3-187
   displaying, 3-187
   in programs, 1-78, 1-79
   menu-based applications, 1-79
cylinder, 4-64

D

Darcy friction factor, 3-72
data input, 3-154
data transmission
   closing ports, 3-51
detecting errors, 3-49
   error testing, 3-40
   parity, 3-217
   receiving, 3-268, 3-269
   serial, 3-381
   size of, 3-40
   specifying baud rate, 3-36
   terminating Server mode, 3-117
time-out, 3-324
via Kermit, 3-160, 3-301
via Kermit server, 3-160, 3-303
data transmissions
  opening ports, 3-212

dates
  calculating days between, 3-75
  calculating past or future, 3-74
  displaying, 3-73, 3-356
  setting, 3-73

debugging, 1-47, 1-49, 3-74, 3-201

decomposing vectors, 3-371
defining procedures
  compiled local variables in, 1-15
  local variables in, 1-14
  local variable structures, 1-11

definite loops, 2-3, 2-26, 2-46, 2-47, 2-49, 2-56
  with counters, 2-10, 2-16
delimiters
  « » for programs, 1-1
declared variables
  specifying in matrices, 3-389
  specifying in plots, 3-81
  specifying in statistical data, 3-57
  summation of squares, 3-389
  summations of, 3-388
dialog boxes (input forms), 3-152
dimensions
  converting, 3-65
  of PICT, 3-223
diodes, 4-69
directories
  changing, 3-142, 3-363
  clearing, 3-53
  creating, 3-69
  current, 3-142
  HOME, 3-142

paths, 3-219
purging, 3-225
display
  area numbers, 1-59
  clearing, 1-74
  clearing stack, 3-51
  creating graphics objects from, 3-163
  freezing, 1-58, 3-123
  printing, 3-237
display mode
  setting, 3-103, 3-117, 3-299, 3-322
  "do" looping, 1-36, 2-19, 2-23, 2-43, 3-89, 3-362
drag force, 4-31

E
editing
  programs, 1-9
eigenvalues
  of matrices, 3-100, 3-101
eigenvectors
  of matrices, 3-100
elastic buckling, 4-3
elastic collisions, 4-30
electricity, 4-9
electrostatic force, 4-11
ellipse, 4-59
energy, 4-15, 4-31
Equation Library
  references, 4-1, 4-82
  starting, 3-104
  subjects, 4-1
  titles, 4-1
equations
  defining sets, 3-193
  disassembling, 3-104
  expanding, 3-109
  least squares solution, 3-178
  rearranging, 3-104
recalling, 3-259
reordering sets, 3-194
retitling sets, 3-194
sets, 3-194
solving linear systems, 3-178
solving quadratic, 3-253
solving sets, 3-197
testing for linearity, 3-168
equation sets
changing titles, 3-194
defining, 3-193
reordering, 3-194
solving, 3-197
errors
actions in programs, 1-51
analyzing, 1-51
causes, 1-51
causing, 1-51, 3-90
clearing last, 1-51, 3-106
conditional structures, 1-53, 1-54, 3-146
detecting in transmission, 3-49
display messages, 1-51
Kermit, 3-159
numbers for, 1-51, 3-106
recalling messages, 1-51, 3-105, 3-159
trapping, 1-53, 1-54, 3-146
user-defined, 1-51
error-trapping, 2-29
error trapping, 2-9, 2-10
escape velocity, 4-49
evaluation
of local variables, 1-13
of test clauses, 1-21, 1-22, 1-36, 1-38
example program
UNMIX, G-5
example programs
animating graphics, 2-45, 2-47, 2-56
applying programs repeatedly, 2-19
Bessel functions, 2-43
calculating median, 2-14
converting from algebraic to RPN, 2-40
converting plots to grobs, 2-45
custom menus, 1-80
displaying binary integers, 2-10
execution times, 2-5
Fibonacci numbers, 2-2
input forms, 1-67
input routines, 1-54, 1-57, 1-60, 1-64, 1-66
inverse functions, 2-54
manipulating math curves, 2-46
mass of an object, 1-79
maximum and minimum elements, 2-22
percentile of a list, 2-14
phone list, 1-67
plotting pie charts, 2-49
preserving calculator status, 2-8
rearranging algebraics, 2-19
right-justifying strings, 2-7
summations, 1-41
surface area of a torus, 1-45
system flags, 1-43
Taylor's polynomials, 2-45
trace mode, 2-53
using conditionals, 1-23, 1-24, 1-25, 1-26
using loops, 1-29, 1-31, 1-33, 1-35, 1-37, 1-40
verifying arguments, 2-36
volume of a sphere, 1-5, 1-6
volume of a spherical cap, 1-6, 1-9, 1-12
Examples subdirectory
creating, 3-345
removing, 3-52
exponents, 3-383
expressions
creating from arguments, 3-15
pattern replacement, 3-180, 3-181
rewriting, 3-180, 3-181
extrapolation, 3-234, 3-235, 3-236

F
factorials, 3-112
factor units, 3-361
false (test result), 1-17, 1-19
Fanning friction factor, 3-113
F distribution, 3-364
Fibonacci numbers, 2-2
flags
annunciators, 1-42
binary integer form, 1-44
clearing, 1-42, 3-45, 3-114, 3-125
control behavior, 1-42
controlling logic with, 2-23, 2-26
default states, C-1
preserving and restoring status, 2-8, 2-9, 2-49
program control, 1-42
recalling, 3-262
recalling states, 1-44
restoring states, 1-44
setting, 1-42, 2-10, 2-23, 2-26, 2-53, 3-304, 3-327
storing states, 1-44
system, 1-42, 1-44, C-1
testing, 1-42, 2-23, 2-26, 3-114, 3-124, 3-125
testing and clearing, 3-114, 3-125
types, 1-42
user, 1-42, 1-44
fluid flows
Darcy friction factor, 3-72
Fanning friction factor, 3-113
fluids, 4-22
focal length, 4-50
force, 4-11, 4-27, 4-44
“for” looping, 1-32, 1-34, 2-10, 2-16, 2-26, 2-46, 2-47, 2-49, 2-56, 3-119, 3-201
formatting
ports, 3-228
Fourier transforms
inverse, 3-147
of arrays, 3-115, 3-147
free fall motion, 4-47
freeing merged memory, 3-121, 3-122
frequency
resonant, 4-19
friction losses, 4-26
functions
applying to list elements, 3-92, 3-102, 3-203
applying to lists, 3-91
defining, 3-77
G
gamma function, 3-112
gas compressibility correction factor, 3-393
gas-compressibility factor, 4-36
gases, 4-32
geometry, 4-58, 4-63

Index-7
global variables
  action in programs, 1-2
  disadvantages in programs, 1-11
  list of, 3-357
graphics
  creating, 3-17, 3-40, 3-131, 3-166, 3-227
  custom, 2-56
  displaying, 3-249
  environment, 3-131
  Picture environment, 3-227
graphics commands
  parallel processing with, G-3
graphics objects
  animating, 3-14
  creating blank, 3-39
  creating from display, 3-163
  creating from stack, 3-133
  displaying, 3-164
  manipulating, 2-23, 2-46, 2-47, 2-49, 2-56
  superimposing, 3-130, 3-134
gravitation, 4-31, 4-47, 4-48, 4-49

H
HALT annunciator, 1-47, 1-59
halting programs, 1-48, 3-136
harmonic motion, 4-54
head loss, 4-24
heat, 4-37
heat capacity, 4-38
heat transfer, 4-37
histograms, 3-137, 3-139
HMS format
  adding in, 3-139
  converting from, 3-141
  converting to, 3-142
  subtracting in, 3-140
Hooke’s law, 4-30

ideal gases, 4-32
"if" branching, 1-20, 1-21, 1-22, 1-53, 1-54, 2-2, 2-23, 2-26, 2-36, 3-101, 3-144, 3-148, 3-149, 3-346
"iferror" branching, 2-29
implicit variable references, 3-305
indefinite loops, 2-7, 2-19, 2-23
ending, 3-89, 3-102, 3-374
with counters, 2-43
independent variables
  specifying for plotting, 3-151
  specifying in matrices, 3-380
  specifying in statistical data, 3-57
  summation of squares, 3-379
  summations of, 3-379, 3-388
index of refraction, 4-50
inductor, 4-17, 4-20, 4-21
initial value problems, 3-274
  error estimate, 3-276, 3-290
  next step, 3-288
  solution step size, 3-277
  with known partials, 3-286
input
  prompting for, 3-238
input forms
  creating, 3-152
  custom, 1-67, 3-152
  for program input, 1-67
  in programs, 1-67
  resetting, 3-203
  saving initial values, 3-203
input plane
  setting the x range, 3-387
  setting the y range, 3-393
intrinsic density of silicon, 3-305

Index-8
inverses
  calculating, 3-156
  logical, 3-201
  of matrices, 3-156
  storing, 3-309
I/O
  closing ports, 3-51
  IR port, 3-157
  Kermit errors, 3-159
  Kermit transmission, 3-160
  selecting port, 3-157
  serial port, 3-157
isothermal expansion, 4-34

J
  junction field-effect transistors, 4-74

K
  Kermit, 3-212, 3-217, 3-230,
    3-268, 3-269, 3-301, 3-303
  error messages, 3-159
  server, 3-160
  transmission, 3-160
keyboard
  defining user, 3-24, 3-328
  in programs, 1-72, 1-73
  recalling definitions, 3-263
  unassigning user keys, 3-80
key location numbers, 1-73
keys
  defining in user keyboard, 3-24
  testing, 3-159
keystrokes
  as program input, 1-72, 1-73
  waiting for, 3-373
killing programs, 1-47, 1-48, 3-161

L
  last argument
    recalling, 2-10, 3-162, 3-163
  length factor, 4-3
libraries
  attaching, 3-31
  detaching, 3-84
  displaying menus, 3-187
  listing, 3-165
  light, 4-50
  linear mechanics, 4-28
  linear motion, 4-47
  linear structure, 1-20
  lines, 3-166
  list concatenation, G-3
list processing
  programming example, 2-29
lists
  action in programs, 1-2
  adding elements of two, 3-11
  applying commands, functions, or programs to, 3-91, 3-92, 3-102, 3-203, 3-302, 3-333, 3-336
  applying executable object repeatedly, 3-302
  assembling, 3-169
  creating from stack, 3-169
  differences between elements, 3-170
  disassembling, 3-169
  extracting elements, 3-127, 3-129, 3-136, 3-341
  first element, 3-136
  last elements, 3-341
  locating objects in, 3-234
  multiplying elements of, 3-171
  parallel processing, G-1
  product of, 3-171
  replacing elements in, 3-244, 3-246
reversing element order, 3-274
sorting, 2-14, 3-315
sublist position in, 3-203
summing elements of, 3-170
local variables, 2-9
action in programs, 1-2
compiled, 1-15
creating, 1-3, 1-11
evaluating, 1-13, 2-19
exist temporarily, 1-11, 1-12, 1-14
naming, 1-11
nested, 2-43, 2-49
passing between programs, 2-47
storing objects in, 2-19, 2-38
local variable structures
advantages, 1-12
as user-defined functions, 1-16
calculations with, 1-3
create local variables, 1-11
defining procedure, 1-11, 1-14
entering, 1-11
operation, 1-3, 1-11
program element, 1-3
syntax, 1-3, 1-11
logic
controlling, 2-26
controlling with flags, 2-23, 2-26
functions, 2-23, 2-26, 2-36, 2-38
logical functions, 1-17, 1-19, 3-13, 3-201, 3-213, 3-382
longitudinal waves, 4-81
loop structures
counters, 1-29, 1-31, 1-33, 1-35, 1-39, 3-323
“do” looping, 1-36, 3-89
“for” looping, 1-32, 1-34, 3-119, 3-201, 3-323
indefinite, 1-27, 1-36, 2-7, 2-19, 2-23, 3-374
keystroke input, 1-73
negative steps, 1-31, 1-35
program element, 1-3
“start” looping, 1-28, 1-30, 3-201, 3-321
summation alternative, 1-40
test commands in, 1-36, 1-38
“while” looping, 1-38
lowercase letters
in names, 1-11
M
Mach number, 4-35
magnetic field, 4-15, 4-43, 4-44, 4-45
magnetism, 4-43
magnification, 4-50
mantissas, 3-179
mass
related to energy, 4-31
mathematical data
plotting, 3-94
matrices
adding rows, 3-283
condition number, 3-60
conjugates of, 3-353
converting to columns, 3-54
converting to rows, 3-282
creating from columns, 3-56
creating from rows, 3-284
deleting rows, 3-284
determinants, 3-82
eigenvalues, 3-101
eigenvalues and eigenvectors, 3-100
electing diagonals, 3-85
identity, 3-143
inverting, 3-156
least squares solution, 3-178
LQ factorization, 3-176
LU decomposition, 3-179
multiplying rows by a constant, 3-260
QR factorization, 3-253
random element, 3-257
rank of, 3-257
reduced row echelon form, 3-286
Schur decomposition, 3-298
singular value decomposition, 3-337
singular values of, 3-338
spectral radius, 3-317
squaring, 3-316
sum of diagonal elements, 3-351
swapping rows, 3-292
transposing, 3-353
mechanics, 4-28, 4-29
memory
checking available, 3-186
freeing merged, 3-121
merging RAM card, 3-190
menu-based applications, 1-79
menu descriptions
PRG BRCH, 1-20, 1-27
PRG RUN, 1-49
PRG TEST, 1-18
menus
custom, 1-78, 1-79, 2-23, 3-187
defining, 3-187
delayed display, 1-72, 1-78
displaying, 3-187
displaying in programs, 1-72, 1-77, 1-79
for libraries, 1-77
for program input, 1-79
last menu, 1-78
numbers for, 1-77, 1-78, 3-264
pages in, 1-78
programmatic uses, 1-77
recalling, 3-264
recalling numbers, 1-78
resuming programs, 1-79
running programs, 1-79
temporary, 2-49, 3-350
message boxes
creating, 3-196
custom, 1-77, 3-196
in programs, 1-77
messages, A-1-17
displaying, 3-88
prompting, 1-56
meta-objects, 2-29
Minhunt game, 3-192
cheating, 3-192
storing, 3-192
mode names
Algebraic-entry, 1-63
Algebraic/Program-entry, 1-9, 1-63
Program-entry, 1-4, 1-9
modes
program entry, 1-4, 1-9
setting, 1-9
Mohr's circle, 4-79
motion, 4-46
names
action in programs, 1-2
negatives, 3-199
nested structures, 2-40, 2-43
new commands, E-1-7
newlines, 1-4, 1-59
NMOS transistors, 4-71
nonideal hydrocarbon gas, 3-393
normal distribution, 3-365
NPN bipolar transistors, 4-73
number bases
  converting between, 2-32
numbers
  action in programs, 1-2
  complex, 3-19
  complex conjugates, 3-62
  disassembling complex, 3-71
  fractional part, 3-121
  imaginary parts, 3-150
  integer part, 3-157
  largest available, 3-183
  rational form, 3-251
  rational form with π, 3-251
  real parts, 3-267
  real to binary, 3-292
  real to complex, 3-293
  rounding, 3-279
  rounding to integer, 3-44, 3-118
  smallest available, 3-193
  squaring, 3-316
  evaluating symbolic, 3-207
  operating system, 3-339
  printing, 3-241
  recalling, 3-261
  removing from stack, 3-95, 3-96
  removing labels from, 3-97
  removing pointers to, 3-200
  replacing a portion of, 3-270
  sign of, 3-306
  size of, 3-41, 3-309
  storing, 3-325
  storing in reserved variables, 3-324
  storing objects in, 3-325
  testing types, 1-20
  truncating, 3-353
  type numbers, 1-20
  type numbers of, 3-359
  unevaluated, 3-254
  object type numbers, 1-20
  Ohm's law, 4-11
  optics, 4-50
  oscillations, 4-54
  output
    labeling, 2-5

O
objects
  actions in programs, 1-2
  backup, 3-248
  comparing, 3-294
  conjugates, 3-300
  converting dimensions, 3-65
  copying, 3-200, 3-226
  decomposing, 3-209
  displaying, 3-88
  duplicating, 3-97, 3-98
  entering in programs, 1-4
  evaluating, 3-107
  evaluating by addresses, 3-339

P
packets
  sending, 3-230
  parallel addition, G-3
  parallelepiped, 4-65
  parallel processing, G-1
  DOLIST, G-2, G-6
  multiple-result commands, G-5
  parity
    setting, 3-217
  pendulum, 4-56, 4-57
  percent functions, 3-47, 3-339
  permutations, 3-224
Index-12
phase delay, 4-16

PICT, 3-226

clearing, 3-105
editing, 3-17, 3-40, 3-166
specifying coordinates, 3-231
superimposing grobs onto, 3-130, 3-134

Picture environment
selecting, 3-131, 3-227

pie charts, 2-49

pixels
checking if on, 3-229
coordinates, 3-71, 3-250
toggling, 3-349
turning off, 3-228
turning on, 3-229

plane geometry, 4-58

plots
3D, 3-207, 3-208
3D perspective, 3-111
autoscaling, 3-32
center, 3-44
changing horizontal width, 3-372
controlling axes, 3-33
creating, 3-36
drawing, 3-94, 3-139, 3-296
drawing axes, 3-95
labeling axes, 3-162
mathematical data, 3-94
resolution, 3-207, 3-208, 3-271
scaling, 3-135, 3-299
setting axes tick-marks, 3-30
setting types, 3-34, 3-61, 3-86, 3-125, 3-132, 3-137, 3-215, 3-296, 3-297, 3-355, 3-375, 3-391
simultaneous, 3-307
specifying dependent variable, 3-81
specifying eye point, 3-111
specifying x-axis display range, 3-384
specifying y-axis display range, 3-390
statistical data, 3-36, 3-94, 3-137, 3-139

plot types
setting, 3-218, 3-221, 3-312
specifying, 3-232

plug-in cards
initializing, 3-228
PN step-junction devices, 4-69
polarization, 4-52
polygon, 4-61
polynomials
evaluating, 3-224
monic, 3-220
roots, 3-238
Taylor's, 3-343
polytropic processes, 4-34
population covariance, 3-222
population standard deviation, 3-242
population variance, 3-247
port
opening, 3-212
selecting, 3-157
ports
closing, 3-51
initializing, 3-228
pressure
hydrostatic, 4-23
PRG annunciator, 1-4, 1-9
PRG BRCH menu, 1-20, 1-27
PRG RUN menu, 1-49
PRG TEST menu, 1-18
print buffer
printing, 3-68
printing
HP 82240A, 3-211
print buffer, 3-68
setting delay, 3-79
trace mode, 2-53
Program Development Link, 1-10, 2-1
programs included, 2-1
Program-entry mode, 1-4, 1-9
program examples
  arbitrary number bases, 2-32
programming techniques
  applied list processing, 2-29
  "case" branches, 2-38, 2-49
  case structures, 2-38, 2-49
  controlling logic with flags, 2-23, 2-26
  custom graphics, 2-56
  custom menus, 2-23, 2-26
  definite loops, 2-3, 2-26, 2-46, 2-47, 2-49, 2-56
  definite loops with counters, 2-10, 2-16
  "do" loops, 2-19, 2-23, 2-43
  error-trapping, 2-29
  error trapping, 2-9, 2-10
  evaluating local variables, 2-19
  "for" loops, 2-10, 2-16, 2-26, 2-46, 2-47, 2-49, 2-56
  "if" branches, 2-23, 2-26
  indefinite looping, 2-33
  indefinite loops, 2-7, 2-19, 2-23
  indefinite loops with counters, 2-43
  interpolation, 2-14
  labeling output, 2-5
  list concatenation, 2-40
  local variables, 2-9, 2-38, 2-43, 2-47, 2-49
  logical functions, 2-23, 2-26, 2-36, 2-38
  logic control, 2-26
manipulating grobs, 2-23, 2-46, 2-47, 2-49, 2-56
meta-object manipulation, 2-29
nested conditionals, 2-23, 2-26, 2-36
nested structures, 2-40, 2-43
object type-checking, 2-40
plot commands, 2-45, 2-46, 2-49
preserving flag status, 2-9, 2-49
programs as arguments, 2-10, 2-19, 2-54
recursion, 2-2, 2-40
restoring flag status, 2-9, 2-49
restoring last argument, 2-10
root-finder, 2-54
setting flags, 2-10, 2-23, 2-26, 2-53
simulating new object types, 2-29
sorting array elements, 2-23
sorting lists, 2-14
"start" loops, 2-14
string and character manipulation, 2-33
string operations, 2-7
structures, 2-5
subroutines, 2-5, 2-10, 2-21, 2-38
tagged output, 2-33
temporary menus, 2-49
testing flags, 2-23, 2-26
using arrays, 2-16, 2-49
using calculator clock, 2-5
using flags, 2-29
using other programs, 2-5, 2-10, 2-21, 2-38
using statistics commands, 2-49
utility programs, 2-38
vectored enter, 2-53
“while” loops, 2-7
programs
  actions for object types, 1-2
  applying to elements of a matrix, 2-29
  applying to list elements, 3-92, 3-102, 3-203
  applying to lists, 3-91
  are sequences of objects, 1-1
  beeping, 1-71
  calculation styles, 1-3
  cancelling halted, 3-161
  causing errors, 1-51
  checksums, 2-1
  comments in, 1-10
  conditional structures, 1-20, 1-53, 1-54
  creating on computer, 1-10
  cursor position during input, 1-63
  debugging, 1-47, 3-74, 3-201
  default input, 1-60
  displaying input forms, 1-67
  displaying menus, 1-72, 1-77, 1-79, 2-23, 2-26, 2-49
  displaying output, 1-74, 1-75, 1-76
  displaying string output, 1-75
  editing, 1-9
  elapsed time, 2-5
  entering, 1-4
  entry modes, 1-4, 1-9
  entry modes during input, 1-63
  error actions, 1-51
  evaluating local variables, 1-13
executing, 1-5
finding roots in, 2-54
flags in, 1-42
getting input, 1-55, 1-56, 1-58, 1-60, 1-72, 1-73, 3-154
HALT annunciator, 1-47
halting, 1-48, 3-136
in local variable structure, 1-3, 1-11
input as strings, 1-60
input forms, 3-152
introduction, 1-1
killing, 1-47, 1-48, 3-161
labeling output, 1-74, 1-75
local variable structures, 1-3, 1-11
loop structures, 1-27, 3-89, 3-374
naming, 1-5
newlines in, 1-4
not evaluating local variables, 1-13
not executing in programs, 1-2
objects in, 1-2
on the stack, 1-4
pausing, 3-373
pausing for output, 1-76
prompting, 1-56, 1-58, 1-60, 3-154
recursion, 2-2
resuming, 1-47, 1-48, 1-56, 1-59, 1-79, 1-82, 3-64
samples, 3-345
scope of local variables in, 1-14, 1-15
single-step execution, 1-47, 1-48, 1-49, 3-320
size of, 2-1
stepping through, 3-201
stopping, 1-5, 3-102, 3-161
storing, 1-5
structures in, 1-3
subroutines, 1-45
test commands, 1-17
trapping errors, 1-53, 1-54, 3-102
turning off calculator, 1-82
user-defined functions, 1-16
using as arguments, 2-10, 2-19, 2-54
using as subroutines, 2-5, 2-10, 2-21, 2-38
utility, 2-38
verifying, 2-1
verifying input, 1-63, 2-36
viewing, 1-9
waiting for keystrokes, 1-72, 1-73, 3-373
projectile motion, 4-48
prompting, 1-56, 1-58, 1-60
prompts, 3-154
Q
quadratic equations
solving, 3-253
quality factor, 4-19
R
RAM
checking available, 3-186
RAM cards
freeing, 3-121, 3-122
merging, 3-190
random numbers
generating, 3-256
in matrices, 3-257
seeding, 3-267
reactance, 4-16
real numbers
converting to binary, 3-292
converting to complex, 3-293
manipulating, 2-33
recalling
flag states, 1-44
last arguments, 3-163
menu numbers, 1-78
rectangle, 4-60
recursion, 2-2, 2-40
reduced row echelon, 3-286
reflection, 4-52
refraction, 4-50
regression
calculating, 3-177
formula used, 3-166
power, 3-250
setting type, 3-37, 3-110, 3-167, 3-176
remainders, 3-195
reserved variables
storing objects in, 3-324
resistance
wire, 4-13
resonant frequency, 4-19
ring, 4-61
root-finder
in programs, 2-54
roots
finding, 3-281
in programs, 2-54, 3-385
of polynomials, 3-238
row operations
adding rows, 3-283
converting rows to a matrix, 3-284
deleting rows, 3-284
multiplying and adding to another row, 3-260
multiplying by a constant, 3-260

Index-16
norms, 3-280
swapping rows, 3-292
RPN syntax
   converting to, 2-40
Runge-Kutta-Fehlberg, 3-274
RUN menu, 1-49

S
   sample standard deviation, 3-300
   sample variance, 3-367
   Schur decomposition, 3-298
   sequential calculations, 3-302, 3-336
   serial input buffer, 3-318
   serial interrupt, 3-295
   serial transmissions, 3-381
      input buffer, 3-318
      interrupting, 3-295
   Server mode
      terminating, 3-117
   silicon
      intrinsic density of, 3-305
   silicon devices, 4-67
   single-step execution, 1-47, 1-48, 1-49
   sinusoidal signal, 4-21
   SI units, 3-360
size
   binary wordsize, 3-265
   initial value solution step, 3-277
   of data transmissions, 3-40
   of objects, 3-41, 3-309
   of programs, 2-1
   of stack, 3-82
software
   version and date, 3-368
   solenoid, 4-20, 4-44
   solid geometry, 4-63
   solid state devices, 4-67
solver
   starting, 3-314
sound waves, 4-81
spectral norm, 3-314
spectral radius of a matrix, 3-317
sphere, 4-66
spring, 4-30, 4-55
squaring, 3-316
stack
   calculations on, 1-3
   clearing, 3-50
   displaying, 3-345
   duplicating objects in, 3-97, 3-98, 3-215
   manipulating, 3-280, 3-281, 3-282, 3-338
   printing, 3-239, 3-240, 3-241
   removing objects from, 3-95, 3-96
   selecting objects from, 3-226
   size of, 3-82
stack display
   clearing, 3-51
stack elements to vectors, 3-370
stack syntax
   in local variable structures, 1-4
   test commands, 1-17
standard deviation
   population, 3-242
   sample, 3-300
   “start” looping, 1-28, 1-30, 2-3, 3-201
state change, 4-33, 4-36
statistical data
   chi-square distribution, 3-363
   clearing, 3-52
   correlation, 3-65, 3-177
   covariance, 3-222
   extrapolating X, 3-235
extrapolating Y, 3-234, 3-236
F distribution, 3-364
maxima, 3-184
mean, 3-185, 3-198
minima, 3-194
normal distribution, 3-198, 3-365
plotting, 3-36, 3-94, 3-137, 3-139, 3-296
population covariance, 3-222
population standard deviation, 3-242
population variance, 3-247
recalling, 3-264
regression, 3-166, 3-177
sample covariance, 3-67
sample standard deviation, 3-300
sample variance, 3-367
sorting by frequency, 3-38
specifying dependent variable, 3-389
specifying independent and dependent variables, 3-57
specifying independent variable, 3-380
stored in $\Sigma DAT$, 3-208
storing, 3-332
summing, 3-351
summing dependent variables, 3-388
summing independent variables, 3-379
summing products of variables, 3-388
summing squared dependent variable, 3-389
summing squared independent variables, 3-379
$t$ distribution, 3-365
upper chi-square distribution, 3-363
upper normal distribution, 3-365
upper Snedecor's $F$ distribution, 3-364
upper students $t$ distribution, 3-365
variance, 3-198
step junction, 4-69, 4-74
storing
flag states, 1-44
programs, 1-5
strain, 4-76
stress, 4-1, 4-76
strings
action in programs, 1-2
as program output, 1-75
concatenation, 2-33
evaluating, 3-333
first characters, 3-204
input converted to, 1-60, 3-334
locating elements in, 3-234
manipulating, 2-7
subdirectories
clearing, 3-52
sublists
number used with DOSUBS, 3-102
subroutines, 2-5, 2-10, 2-21, 2-38
debugging, 1-49
in programs, 1-45
operation, 1-45
single-step execution, 1-49, 3-320
summations
alternative to looping, 1-40
of dependent variables, 3-388
of independent variables, 3-379
of products of statistical
variables, 3-388
of squared dependent
variables, 3-389
of squared independent
variables, 3-379
of statistical data, 3-351
symbolic arrays, 2-29
symbolic constants, 3-99, 3-143,
3-183, 3-193
evaluating, 3-63
symbolic objects
evaluating, 3-207
system time, 3-346, 3-347
setting, 3-347

T
tagged objects
as program output, 1-74
creating, 3-341
Taylor’s polynomials
graphing, 2-45
t distribution, 3-365
temperature
change, 3-344
increment, 3-348
terminal velocity, 4-49
test commands
algebraic syntax, 1-17
combining results, 1-19
comparison functions, 1-17
flag tests, 1-42
in conditional structures,
1-17, 1-20
in loop structures, 1-36, 1-38
logical functions, 1-19
results of, 1-17, 1-18, 3-159
stack syntax, 1-17

testing
algebraics, 1-19
binary integers, 1-19
flag states, 1-42
testing linear structure, 1-20
TEST menu, 1-18
thermal expansion, 4-39
tick-marks
setting in plots, 3-30
time, 3-347
and date, 3-356
setting, 3-347
time-out
during data transmission,
3-324
toroid, 4-21, 4-45
trace mode, 2-53
transistors, 4-67
transverse waves, 4-80
trapping errors, 1-51
triangle, 4-62
true (test result), 1-17, 1-19
turning off the calculator, 3-211
TVM, 3-357
begin mode, 3-358
end mode, 3-358
solving, 3-358

U
units
converting angular, 3-99,
3-294
creating from stack, 3-362
factoring, 3-361
numeric portion, 3-366
SI, 3-360
upper chi-square distribution,
3-363
upper normal distribution,
3-365
upper Snedecor’s F distribution, 3-364
upper students t distribution, 3-365
user-defined errors, 1-51
user-defined functions
   internal structure, 1-16
user keyboard
   defining keys, 3-24
user keys
   assigning, 3-24
   unassigning, 3-80
utility programs, 2-38

V
variables
   action in programs, 1-2
   adding objects to, 3-329
   clearing, 3-52, 3-53
   decrementing, 1-39, 3-76
   defining, 3-77
   dependent, 3-57, 3-388, 3-389
   designated as calculated, 3-185
   designating user-defined, 3-197
   dividing by, 3-331
   incrementing, 1-39, 3-150
   independent, 3-57, 3-151, 3-379, 3-380, 3-388
   isolating, 3-158
   list of currently used, 3-367
   list of particular type, 3-357
   multiplying, 3-330
   negating, 3-313
   ordering, 3-214
   picture, 3-226
   port, 3-248
   printing, 3-240
   purging, 3-243
   showing, 3-305
   solving multiple, 3-195
   storing objects in, 3-325
   subtracting objects from, 3-330
   types, 3-368
   variable types, 3-357
   variance, 3-198
      population, 3-247
      sample, 3-367
   vectored enter, 2-53
   vectors
      converting from matrices, 3-54
      converting to elements, 3-54
      creating from elements, 3-56
      creating from stack, 3-369
      from stack elements, 3-370
      to stack elements, 3-371
   vibrations, 4-80
   view volume
      setting the depth, 3-392
      setting the height, 3-394
      setting the width, 3-386
   voltage, 4-9, 4-67
   Vroom, Fruit of the, 2-52

W
waiting
   displaying output, 1-76
      for keystrokes, 1-72, 1-73
   warmstart log, 3-377
   waves, 4-80
   “while” looping, 1-38, 2-7, 3-270, 3-374
   while loops, 3-374
   wordsize (binary)
      testing, 1-19
   wordsize of binary integers, 3-335

Index-20
X
x-axis
  specifying display range, 3-384
Xmodem
  receiving objects, 3-384
  sending objects, 3-386

Y
y-axis
  specifying display range, 3-390
Contacting Hewlett-Packard

For Information about Using the Calculator. If you have questions about how to use the calculator, that are not covered in this manual, first check the table of contents, the subject index, and “Answers to Common Questions” in appendix A of the HP 48 User’s Guide. If you can’t find an answer in either manual, you can contact the Calculator Support Department:

Hewlett-Packard
Calculator Support
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.
(503) 715-2004 (Mon.–Fri., 8:00am–3:00pm Pacific time)
(503) 715-3628 FAX

For Hardware Service. See appendix A of the HP 48 User’s Guide for diagnostic instructions and information on obtaining service. But, before you send your unit for service, please call HP Calculator Support at the number listed below.

Hewlett-Packard
Corvallis Service Center
1030 N.E. Circle Blvd., Bldg. 11
Corvallis, OR 97330, U.S.A.
(503) 715-2004 (HP Calculator Support)

If you are outside the United States, see appendix A of the HP 48 User’s Guide for information on locating the nearest service center.

HP Electronic Information Service. This service provides information on HP 48 calculators and is free of charge—you pay only for the phone call or Internet service. There are two ways to connect:

- **via modem**: (503)715-4448. It operates at 2400/9600/14400 baud, full duplex, no parity, 8 bits, 1 stop bit.
- **via the Internet**: Access as hpcvbbs.external.hp.com (192.6.221.13) using telnet, ftp, or World-Wide-Web browser (http://hpcvbbs.external.hp.com/hp48g.html).
Contents

1: Programming
2: Programming Examples
3: Command Reference
4: Equation Reference

A: Error and Status Messages
B: Table of Units
C: System Flags
D: Reserved Variables
E: New Commands in the HP 48G Series
F: Technical Reference
G: Parallel Processing with Lists