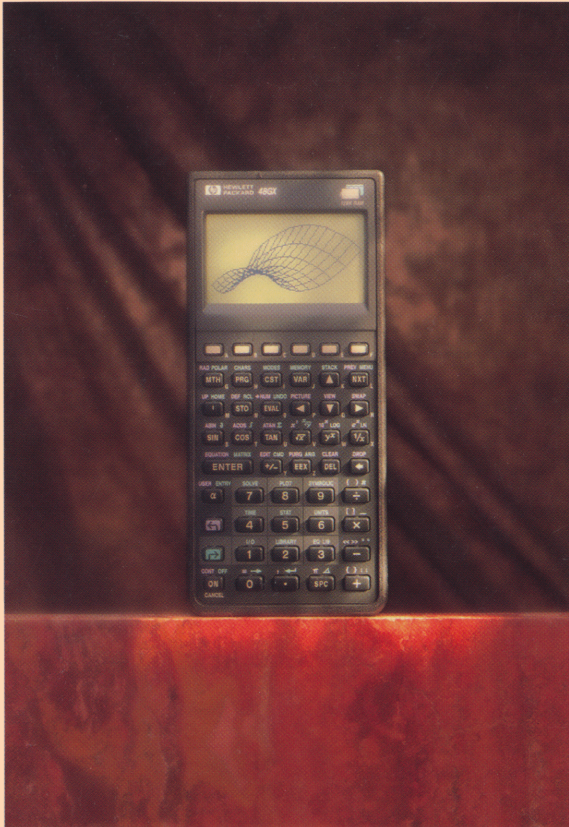


The HP 48 Handbook



James Donnelly

The HP 48 Handbook

2nd Edition

James Donnelly

Copyright © James Donnelly 1990–1994

All rights reserved. No part of this book may be reproduced, transmitted, or stored in a retrieval system in any form or by any process, electronic, mechanical, photocopying, or means yet to be invented, without specific prior written permission of the author.

Second Edition

First Printing, June 1993

Second Printing, October 1994

ISBN 1–879828–04–9



Armstrong Publishing Company
1050 Springhill Drive
Albany OR 97321 USA

Preface to the 2nd Edition

The Hewlett-Packard HP 48SX calculator was introduced in March, 1990 to an enthusiastic reception by students and engineers around the world. In April 1991 a second model, the HP 48S, was introduced at a lower cost, making the power of the HP 48 family available to a wider audience.

In June 1993 Hewlett-Packard announced a major upgrade to both models of the HP 48, extending the built-in command set and providing new organization to the most used tasks with focused user interfaces. Along with the new software, the HP 48GX has 128K of random access memory (up from 32K), while the HP 48G remains at 32K.

The HP 48 Handbook has been used by many HP 48 owners to provide a reference to the calculator and to the HP 82211 HP Solve Equation Library Card. This new edition of the Handbook has been designed to serve owners of both the original and new models of the HP 48 family. HP 48 fans who are upgrading from the original model to the new model will notice that in addition to the new functions and user interfaces, certain areas, such as memory organization, are somewhat different. Compatibility issues between new and old models are discussed, and many ideas are provided to help you get the most out of your new machine. The Example Programs and System Programming chapters are new, and provide exciting new programs and tools to use.

Acknowledgements

This second edition includes changes and improvements that originate from both reader feedback and HP's enhancements of the HP 48 calculator. Special thanks go to Lee Buck, Joseph K. Horn, Wlodek A.C. Mier-Jedrzejowicz, Jeremy Smith, and Eric L. Vogel for many thoughtful comments, suggestions, and hours of hard work.

Lee Buck contributed the DSORT program for sorting directories and the date utility programs. The program TANK is based on a program of the same name originally written by Doug Cannon. Joseph K. Horn contributed the PFIT, SHWP, and SQPQ programs. The program TRAIN was contributed by Jeremy Smith. The program TREE is based on a program written by Charles Talbot, who credited the book "Fractal Programming in C" by Roger T. Stevens as his source.

Ted Beers, Ron Brooks, Sharon Butterfield, Diana Byrne, Dan Coffin, Tom Diamond, Russ Donnelly, Grant Garner, Richard Nelson, Charlie Patton, Scott Rohrer, Bill Wickes, Bob Worsley, and Dennis York contributed advice, ideas, support, and encouragement.

The cover photograph is by Peter Krupp.

Three outstanding books contributed to the creation of some of the example programs relating to date utilities, Brownian motion, chaos, and fractal images. All three are highly recommended:

Duffet-Smith, Peter *Practical Astronomy With Your Calculator*, 3rd Ed. (Cambridge University Press, 1988)
ISBN 0-521-35699-7

Lauwerier, Hans *Fractals - Endlessly Repeated Geometrical Figures* (Princeton, New Jersey: Princeton University Press, 1991)
ISBN 0-691-02445-6

Gulick, Denny *Encounters With Chaos* (McGraw-Hill, Inc., 1992)
ISBN 0-07-025203-3

Last, but not least, this 2nd edition would have been utterly impossible were it not for the support of my beloved wife Janet.

To Russell and Marian Donnelly

Contents

Introduction	1
Objects, Names, and Constants.....	3
Object Types.....	4
Real and Complex Numbers.....	5
Binary Integers	6
Unit Objects	7
Backup Objects.....	8
Library Objects.....	9
Object Sizes.....	11
Object Evaluation	12
Operator Precedence.....	13
Variable Names	14
Symbolic Constants.....	15
Memory	16
Memory Organization	16
User Memory.....	17
Temporary Memory	18
Configuring RAM Cards.....	22
Graphics	24
Graphics Coordinates	24
Graphics Operations	26
Converting Text to Graphics	29
Animation Sequences.....	31
Stack View Program.....	33
Graphics Object Structure	34
PPAR.....	36
VPAR	39
Programming.....	40
Program Execution.....	40
Single-Stepping Programs.....	41
Local Variables.....	42
Programs as Subroutines.....	45
User-Defined Functions	48
Looping Structures	49
Conditional Structures.....	52
Error Trapping.....	54
Data Entry.....	56
Displaying Results.....	66
Recursion	69
List Processing	70
Meta-Objects	73

Example Programs.....	80
Greatest Common Divisor.....	80
Least Common Multiple	81
Square Root's Partial Quotients	81
Polynomial Curve Fitting	83
Slope of a Line	86
User Interface for User-Defined Functions.....	87
Sorting Directories	89
Date Utilities.....	90
Status Line Animation	95
Customizing the Solver	99
Amortization Plot	109
Pascal's Triangle.....	111
Plotting Inequalities	113
Brownian Motion	116
Entropy	119
Chaos.....	123
Fractal Trees.....	131
Julia Sets.....	135
The Mandelbrot Set.....	137
System Programming.....	139
User-RPL vs. System-RPL.....	139
SYSEVAL & Version Identification.....	140
The Dangers & Benefits of System-RPL	141
Assumptions in This Chapter	141
Fixed Entry Points.....	141
Examples in this Chapter	142
Naming Conventions.....	143
Checking Arguments.....	144
Binary Integers	145
Character Constants	150
Character Strings.....	151
Real and Extended Real Numbers.....	157
Composite Objects	168
Tests	172
Stack Operations	176
Graphics Operations	180
Keyboard Control.....	192
Streamlining Finished Programs.....	195
LIBEVAL.....	197
HP Tools.....	200
System Operations.....	201
Invoking System Operations	201
System Halt Log.....	202
Interactive Self Test	203

Statistics Data	204
Σ DAT.....	204
Σ PAR	205
Character Codes.....	206
Data Transfer.....	208
Data Transfer Methods.....	208
Kermit Protocol	209
Backing Up the HP 48	211
Restoring the HP 48	212
ASCII File Transfer	213
Character Translations	214
IOPAR.....	215
Cables.....	216
Printer Control.....	217
PRTPAR.....	219
Messages.....	220
Menus	228
Custom Menus	228
Built-In Menus	230
User-Defined Keys.....	233
Setting User Mode.....	233
Key Locations	233
Standard Keys	233
Flags.....	234
Built-In Units	237
Equation Library Reference.....	242
Equation Reference	242
Constants Reference.....	247
Command Index	249
Command Reference.....	265
Alpha Keyboard	338
Program Index.....	339

Introduction

The HP 48 Handbook is designed with the programmer in mind – a concise combination of system descriptions and detailed reference information. Many example programs are provided to help you along and to stimulate your imagination. A chapter is devoted to *System Programming*, which opens a door into the world “under the hood” of the HP 48. The *Command Reference* is your encyclopedic reference to all commands.

Fundamental Concepts. The HP 48 world revolves around the stack, which is implemented as a dynamically allocated last-in-first-out (LIFO) structure which can hold any number of objects of different sizes and types (see *Objects, Names, and Constants*). All commands take their (zero or more) arguments from the stack and return any results to the stack. For instance, consider the following display:

{ HOME }	
4:	"Janet"
3:	(3,4)
2:	'57+X'
1:	2.47
OVER ROT ROLL ROLLO PICK DEPTH	


Level 1 contains the number 2.47, level 2 the algebraic expression '57+X', level 3 the complex number (3,4), and level 4 the string "Janet". Now execute the multiply function. While multiply is executing, the arguments are removed from levels 1 and 2, leaving (3,4) in level 2, and the string "Janet" in level 3. When the multiplication is complete the result is returned to the top of the stack:

{ HOME }	
4:	
3:	"Janet"
2:	(3,4)
1:	'(57+X)*2.47'
OVER ROT ROLL ROLLO PICK DEPTH	

Many commands are type-sensitive, that is, they perform different operations for different types of input parameters. For the complete descriptions for each command, see *Command Reference*.

Example Programs. There are several example programs and program fragments in this book. Each complete program is named and printed with a size and checksum.

All characters in the programs are case-sensitive. The names of commands are always uppercase. By convention, the names of global variables are uppercase, and of local variables are lowercase. If another example program is used in an example, the name will be shown in **bold** characters.

Verifying Example Programs. While the command line entry of a program may be free form, with the  keystroke being valid between words, graphics objects must be entered exactly as shown, with no extra breaks in the command line when entering the data.

If you enter a program into the HP 48, use the BYTES function to make sure the program in the calculator matches the version in the book.

Computing Checksums. To compute a checksum for a named program, enter the program and store it in a variable with the same name that's in the example. Then, place the name (such as 'VSCAN') on the stack and execute BYTES. The size (including the name) and checksum will be returned to the stack.

To compute a checksum for a program fragment, place a copy of the fragment in level 1 and execute BYTES.

For instance, the program

« DROP SWAP »

is 15 bytes long and has the checksum #5197h.

Objects, Names, and Constants

Object is a general term for anything that can be put on the stack or stored in a variable. Any object may be described in terms of its *type* and *value*. For instance the number 247 has type “real number” with value 247.

Objects may be classified into several broad categories:

- A *data object* contains information, such as a number or a sequence of characters. Real numbers, complex numbers, binary integers, arrays, and strings are examples of data objects.
- A *procedure object* is a collection of objects that perform a task in order. Programs and algebraic expressions are procedure objects, and may be evaluated, placed on the stack or stored in variables just like any other object.
- A *name object* permits an object to be referenced by name.
 - *Global names* refer to corresponding variables that are available at any time. By convention, global variable names are written in uppercase (**A**).
 - *Local names* refer to corresponding local variables that exist only within the scope of the executing program that defines them. By convention, local variable names are written in lowercase (**a**).

In general, objects may be stored in variables or manipulated on the stack regardless of their type. Some HP 48 functions and commands perform different operations based on the type of object supplied as a parameter. For instance, the + function executes differently for strings (concatenates) than for real numbers (adds).

Object Types

Different object types are distinguished in the stack display through their *delimiters* – characters that are unique to that type of object. For instance, strings are surrounded by quote marks ("), and programs are contained in French quotes («»).

Type	Object	Example
0	Real number	1.2345
1	Complex number	(2.3,4.5)
2	String	"ABC"
3	Real array	[1 2 3]
4	Complex array	[(1,2) (3,4)]
5	List	{ "ABC" Var }
6	Global name	X
7	Local name	y
8	Program	« A 2 + »
9	Algebraic	'Y=X^2'
10	Binary integer	# 247d
11	Graphics object	Graphic 131 x 64
12	Tagged object	Dist: 34.45
13	Unit object	32_ft/s^2
14	XLIB name	XLIB 766 1
15	Directory	DIR ... END
16	Library	Library 766: ...
17	Backup object	Backup HOMEDIR
18	Built-in function	SIN
19	Built-in command	SWAP
20	Internal binary integer	<247d>
21	Extended real number	Long Real
22	Extended complex no.	Long Complex
23	Linked array	Linked Array
24	Character object	Character
25	Code object	Code
26	Library data	Library Data
27–30	External objects	External

Some objects may only be manipulated by unnamed internal system objects (see *System Programming*).

Real and Complex Numbers

Real Numbers. Real numbers have a 12-digit mantissa between 1 and 9.9999999999 and a 3-digit exponent between -499 and +499. During math operations, real numbers are expanded to have a 15-digit mantissa and a 5-digit exponent during the calculation, then rounded back to the 12-digit value when returned as results.

Complex Numbers. Complex numbers are represented by pairs of real numbers in parentheses: (2,3) (1.2,5). The Rectangular (x,y) and Polar (r,θ) display modes (flags -15 and -16) control the appearance of a complex number on the stack, but do not affect the internal form. For instance (2,3) is displayed in polar form as (3.60555127546, 456.309932474), but it is still stored internally as (2,3).

Vectors and Matrices. Vectors and matrices may be composed of either real or complex numbers. Some examples:

[1 2]

Real vector

[[1 2]
[3 4]]

Real matrix

[[(1,1) (1,2)
[(2,1) (2,2)]]

Complex matrix

Related Commands: The commands $R \rightarrow C$ and $C \rightarrow R$ convert between real and complex numbers or real and complex arrays. $C \rightarrow R$, $V \rightarrow$, and $OBJ \rightarrow$ decompose a complex number to its real and imaginary parts. $C \rightarrow R$ separates a complex array into an array of real components and an array of imaginary components. $OBJ \rightarrow$ separates a complex array into a series of complex numbers followed by a list containing the dimensions of the original array. If Complex Mode (flag -19) is set, $\rightarrow V2$ creates a complex number instead of a two element array. RE returns the real component of a number or array; IM returns the imaginary component. ARG returns the polar angle θ of a coordinate pair (x,y). $SIGN$ returns a unit vector in the direction of the input argument (x,y).

Binary Integers

Binary integers are entered and displayed with a leading # delimiter and a trailing b, d, h, or o to indicate the base. The trailing character can be omitted if the HP 48 is already set to the desired base mode.

Examples: #101101b #247d #7DACH

The commands STWS and RCWS may be used to store or recall the wordsize, which may be up to 64 bits. The wordsize controls the interpretation of arguments and the results of arithmetic operations. For instance, if a binary integer is added to a real number, the real number is truncated to the current wordsize, and the result is a binary integer truncated to the current wordsize.

Note that changing the wordsize alone does not affect a binary integer – you cannot truncate a binary integer merely by changing and restoring the word size.

Binary Operators. The operators AND, OR, XOR, and NOT are available for logical comparisons. The bit-by-bit comparisons are performed up to the current wordsize according to the following table:

TRUTH TABLE					
arg ₁	arg ₂	arg ₁ AND arg ₂	arg ₁ OR arg ₂	arg ₁ XOR arg ₂	NOT arg ₁
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

Related Commands: The following commands are useful for working with binary integers: B→R, RCWS, RL, RLB, RR, RRB, R→B, SL, SLB, SR, SRB, and STWS.

Unit Objects

Unit objects are entered and displayed in the form: *number_units* where *number* is a real number and *units* is an algebraic expression containing unit names, prefixes, exponents and the operators $*$, $/$, and $^$. (A unit object may only contain one $/$ operator.)

Examples:

`32_ft/s^2`

`Density: 25_g/cm^3`

Units in Menus. Unit objects in built-in menus or custom menus provide three types of functionality:

- Primary keys append the unit on the key to the numerator of the level 1 object.
- Left-shifted keys convert the level 1 object to the unit on the key.
- Right-shifted keys append the unit on the key to the denominator of the level 1 object.

User-Defined Units. A user-defined unit may be created from any combination of the built-in units or other user-defined units. To create a user-defined unit, store the definition in a variable whose name is the name of the new unit.

For example, create the user-defined unit *week* by storing 7_d in the variable *week*. Executing UBASE on 2_week yields 1209600_s. The object 1_week stored in a custom menu will now behave like any other unit-related menu key.

Related Commands: The following commands are useful for working with unit objects: CONVERT, OBJ→, UBASE, UFACT, →UNIT, and UVAL.

Backup Objects

Backup objects are used to store backed-up data in independent memory (ports 1 through 33 for the HP 48G/GX, or ports 1 or 2 for the HP 48S/SX) or in port 0. A backup object may contain any object, including directory structures.

Backup Identifiers. The *contents* of a backup object are referenced by a *backup identifier* (eg: `:1:FRED`), which is a port-tagged name.

The wildcard `&` may be used for the port number for the commands RCL, EVAL, and PURGE. When the wildcard is evaluated, memory is searched in the order of ports 2, 1, 0, and then main memory for the first occurrence of the specified name.

If a backup object contains a directory structure, an object within that directory structure may be recalled or evaluated by specifying the path and name of the object in a port-tagged list.


For instance, `:1:{ EEDIR FRED }` refers to the object FRED in a directory stored in backup object EEDIR in port 1.

Creating Backup Objects. A backup object is created by executing the STO command with the object in level 2, and the port-tagged name in level 1. For instance, the sequence `'FRED' RCL :1:BFRED STO` recalls the contents of variable FRED to the stack and creates a backup object called BFRED in port 1.

Recalling Backup Objects. The contents of a backup object may be recalled in two ways:

- Press $\boxed{\rightarrow}$ **LIBRARY**, `PORT0`, `PORT1`, or `PORT2`, then $\boxed{\rightarrow}$ and the menu key for the backup object.
- Place the backup identifier on the stack and execute RCL.

Evaluating Backup Objects. The contents of a backup object may be evaluated two ways:


- Press  **LIBRARY**, **PORT0**, **PORT1**, or **PORT2** for the port number, then the menu key for the backup object.
- Place the backup identifier on the stack and execute **EVAL**. **EVAL** also accepts a list of backup identifiers.

Purging Backup Objects. To purge a backup object, place the backup identifier on the stack and execute **PURGE**. A backup identifier may be included in a list supplied to **PURGE**.

Related Commands: **PVARS** takes a port number as its argument and returns two results:

- Level 2 contains a list of backup objects and library IDs.
- Level 1 contains the type of memory in the port – "**SYSRAM**", "**ROM**", or a number showing the amount of available independent RAM.

Library Objects

Library objects are collections of one or more objects that generally extend the built-in command set. Libraries are referenced by a *library#* or a library identifier (**! port# library#**), depending on the command. On the HP 48S/SX the title of the library may be displayed by pressing  **REVIEW** in the **LIBRARY** menu.

Installing a Library. Library objects only extend the command set when they are stored in a port (0, 1, or 2) and *attached* to a directory in user memory.

To use a library, perform the following:

- Store the library object in a port, such as port 0. For instance, if the library object is in level 1 of the stack, execute 0 STO.
- Turn the calculator off, then on again. The calculator will perform a system halt, which updates the system configuration to recognize the new library.
- Attach the library to the desired directory.
 - To attach a library to the current directory, enter the *library#* and execute ATTACH.
 - To detach a library from the current directory, enter the *library#* and execute DETACH.

Note: Most commercially produced libraries will automatically attach to the HOME directory. Any number of libraries may be attached to HOME, but only one library may be attached to each subdirectory.

Removing a Library. To purge a library, perform the following:

- Ensure that the library object does not appear on the stack as **Library nnn: ...**. Either store the library in a variable or execute NEWOB to create a unique copy.
- Change to the directory to which the library is attached.
- Enter the library#, such as **:2:272** and execute DETACH.
- Enter the library ID, such as **:2:272** and execute PURGE.

Note: When you detach a library, programs that use commands or functions in that library will now read XLIB nnn mmm when displayed on the stack. You cannot enter a library command into a program in the form XLIB nnn mmm, so you can no longer edit the program. When the program is executed, missing library commands generate the error “Undefined XLIB Name”.

Object Sizes

The following table lists the typical sizes for selected object types. Note that the HP 48 saves temporary memory by using built-in objects for some common values. The most commonly used built-in objects are the real integers from -9 to $+9$, the complex constant $(0,1)$, the real constants (e , π , MINR, and MAXR), and many internal binary integers. When a built-in object is used, only 2.5 bytes are used. For instance, the real number 9 is built into the HP 48, so when you enter 9 and execute BYTES, only 2.5 bytes are used to store a pointer to the built-in number, whereas 10.5 bytes are required to store the number 2.47.

Type	Object	Size (bytes)
0	Real number	10.5
1	Complex number	18.5
2	String	5 + number_of_characters
3	Real array	12.5 + 8 * number_of_elements
4	Complex array	12.5 + 16 * number_of_elements
5	List	5 + size_of_included_objects
6, 7	Unquoted name	3.5 + number_of_characters
6, 7	Quoted name	8.5 + number_of_characters
8	Program	12.5 + size_of_included_objects
9	Algebraic	5 + size_of_included_objects
10	Binary integer	13
11	Graphics object	10 + rows * CEIL(cols/8)
12	Tagged object	3.5 + tag_characters + object_size
13	Unit object	7.5 +
	Real magnitude	10.5 (2.5 if built-in)
	Prefixes	6
	Unit names	5 + number_of_characters
	Operators (*, /, or ^)	2.5
	Exponents	10.5 (2.5 if built-in)
14	XLIB name	5.5
15	Directory	6.5 + size_of_included_objects
17	Backup object	5 + no._of_name_chrs + incl._obj
18, 19	Function or command	2.5
20	Internal binary integer	5
21	Extended real number	13
22	Extended complex no.	23.5
24	Character object	3.5

Object Evaluation

Evaluation of an object may be either implicit or explicit. Objects being entered on the command line, such as a real number or the name of a command such as +, are implicitly evaluated unless surrounding delimiters delay evaluation. An object on the stack may be explicitly evaluated by executing EVAL.

Evaluation results vary with the type of object:

- When a global variable name is evaluated, the contents of the variable are evaluated. To place a global variable name on the stack, enclose it in tick marks ('X').
- When a local variable name is evaluated, the contents of the local variable are recalled to the stack, but *not* evaluated. If a local variable contains a real number, the behavior is essentially the same as for a global variable, but if the local variable contains a program, the program will only be recalled to the stack. You can use a subsequent EVAL to evaluate the program.
- When a program is evaluated, global names are evaluated unless surrounded by ticks ('), the contents of local names are recalled to the stack, commands are executed, and all other objects are put on the stack.
- When an algebraic object is evaluated, the value it represents is computed and returned to the stack. Algebraic objects being evaluated obey rules of precedence – see the table on the next page.
- When a list is evaluated, global names are evaluated, programs are evaluated, commands are executed, and all other objects are placed on the stack.
- All other objects are placed on the stack.

Operator Precedence

Operator precedence controls the order in which calculations take place within an algebraic expression. Functions with the highest precedence (1) are evaluated before those with the lowest precedence (11). The evaluation order is left-to-right for operators having the same precedence. For instance, in the expression ' $3+5*7$ ', the multiply operation takes precedence over the add, resulting in the answer 38, whereas the answer would be 56 if evaluated from left to right without following the rules of precedence.

Level	Operation
1	Expressions within parentheses
2	Functions
3	! (Factorial)
4	Power (^) and square root (√)
5	Negate (−) multiply (*) divide (/)
6	Add (+) and subtract (−)
7	Relational operators (==, ≠, <, >, ≤, ≥)
8	AND and NOT
9	OR and XOR
10	Left argument for ! (where)
11	=

Variable Names

Variable names may contain letters, digits, and most characters. Names may not start with a digit, match a command name, or contain object delimiters or the characters $+$ $-$ $*$ $/$ $^$ $\$$ $=$ $<$ $>$ \leq \geq \neq \div $\sqrt{}$ $!$ space, comma, or $@$. Variable names that begin with \leftarrow are interpreted as local variables (see *Local Variables*).

Reserved Variables. The HP 48 stores information for various commands in *reserved variables*. Reserved variables may reside in any directory, and may be used in more than one directory at a time.

Name	Description
<i>ALRMDAT</i>	Current alarm editing data
<i>CST</i>	Custom menu contents
<i>EQ</i>	Current equation for SOLVE and PLOT
<i>EXPR</i>	Current expression for symbolic operations
<i>IERR</i>	Uncertainty of integration
<i>IOPAR</i>	I/O parameters (HOME directory only)
<i>MHpar</i>	Saves the state of the Minehunt game
<i>Mpar</i>	Multiple Equation Solver equation set
<i>Nmines</i>	Specifies the number of Minehunt mines
<i>PICT</i>	References the graphics display
<i>PPAR</i>	PLOT parameters
<i>PRTPAR</i>	PRINT parameters (HOME directory only)
<i>VPAR</i>	3D PLOT view volume parameters
<i>ZPAR</i>	Stores a copy of <i>PPAR</i> from previous zoom
<i>der...</i>	User-defined derivatives begin with <i>der</i>
<i>n1, n2, ...</i>	Integers created by ISOL
<i>s1, s2, ...</i>	Signs created by ISOL and QUAD
<i>ΣDAT</i>	Current statistical matrix
<i>ΣPAR</i>	Statistics parameters

Notes:

- The I/O setup commands *only* modify the copy of *IOPAR* in the HOME directory.
- The print commands *only* modify the copy of *PRTPAR* in the HOME directory.
- *PICT* is not directory-dependent. It only refers to graphics display memory.
- With the exception of *PICT*, you can store any object into a reserved variable, but the subsequent execution of commands that depend on that reserved variable may be unpredictable or generate an error.
- You may purge a reserved variable to save memory.

Symbolic Constants

The HP 48 has five constants which may be used in symbolic form or as approximate numerical values.

Name	Machine Value
π	3.14159265359
e	2.71828182846
i	(0,1)
MAXR	9.99999999999E499
MINR	1.E-499

System flags -2 and -3 control evaluation of symbolic constants:

Flag	Description	Clear	Set	Default
-2	Symbolic Constants	Symbolic form	Numeric form	Clear
-3	Numeric Results	Symbolic results	Numeric results	Clear

Memory

Memory in HP 48 calculators is accessed in four-bit quantities (nibbles, or 1/2 bytes) within a 20-bit address space, yielding a 512K byte address space. The BYTES command, which returns the size and a checksum for an object, will sometimes show a size such as 106.5, reflecting that the object occupies 213 nibbles of memory.

The HP 48SX and the HP 48GX have two ports which may accomodate plug-in cards containing either random-access memory (RAM) or read-only memory (ROM). The rest of this chapter is devoted to a discussion of system memory organization and possible uses of plug-in cards.

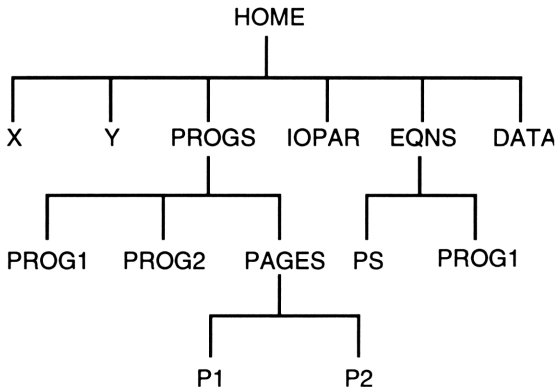
Memory Organization

Memory in the HP 48 is organized as follows:

- | | |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| System ROM | The operating system resides in 512K bytes (256K in the HP 48S/SX) of read only memory (ROM). This command set may be extended through the use of library objects which reside in ROM or RAM (see <i>Library Objects</i>). Half of this memory may be layered with system RAM. |
| System RAM | There are 32K bytes of random access memory (RAM) in the HP 48S, HP 48SX, and the HP 48G. There are 128K bytes of RAM in the HP 48GX. Some memory is reserved for display memory and temporary memory. |
| Plug-in ROM | Plug-in ROM application cards, available from a number of manufacturers, may extend the built-in command set. |
| Plug-in RAM | HP 48SX or HP 48GX RAM may be extended by adding plug-in RAM cards that contain either 32K (HP 82214A) or 128K (HP 82215A). Plug-in RAM may be configured two ways (see <i>Configuring RAM Cards</i>). |

User Memory

User memory may be organized into a tree structure of directory objects, which are implemented as variables stored in the HOME directory.



The status line displays the current directory path, and the VAR menu displays the current directory:



In the example above, the current directory is PAGES, which contains variables P1 and P2.

Creating a Directory. A directory may be created with the command CRDIR. To store variables in the new directory move to the new directory by evaluating its name or pressing the corresponding key in the VAR menu.

Accessing Variables. When a variable name is evaluated, the current directory is searched first. If the variable is not found, its parent directories are searched in ascending order until the variable is found. In the example above, there are two variables named PROG1. Different directories may have variables of the same name.

Changing Directories. To change to a lower directory, simply evaluate its name. To return to the previous level, execute UPDIR (see *Menu Traversal Program*). Evaluating a list that starts with HOME followed by directory names can quickly change the current directory to any other place in user memory. For instance, if the current directory is PAGES, evaluating { HOME EQNS } will change the current directory to EQNS. A port-tagged path may be used for RCL and EVAL, but you must move to the target directory for STO.

Changing a Directory Name. To change the name of a directory or move the directory to another location, perform the following steps:

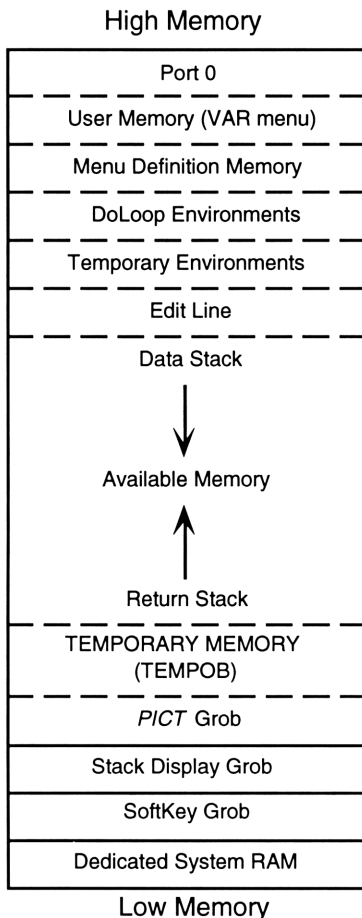
- Recall the directory to the stack
- Purge the old directory
- Move to the new location
- Enter the new name and execute STO.

Purging a Directory. The PURGE and PGDIR commands may be used to purge a directory. The PURGE command only removes empty directories; PGDIR removes a directory and its contents.

Saving User Memory. The commands ARCHIVE and RESTORE may be used to save and recover all of user memory (see *Data Transfer*).

Temporary Memory

The data stack in the HP 48 is actually a stack of pointers which refer to objects elsewhere in memory. Temporary memory is the calculator's "scratchpad". All objects that are not stored in a port or in a user variable reside in temporary memory. Many commands require temporary memory to construct intermediate objects or new objects returned as results to the stack.



Use of Temporary Memory. To understand temporary memory a little more, consider what happens when two math operations are performed. Enter the numbers 1.5 and 2.6 on the stack. These numbers now reside in temporary memory, referred to by pointers on the data stack. When the numbers are added, the result, 4.1, is a number in temporary memory referenced by a pointer in level 1 of the data stack. The objects 1.5 and 2.6 remain in temporary memory, referenced by pointers that save the Last Arguments.

Now add 2.8 to the result in level 1. The level 1 pointer on the data stack refers to the object 6.9 in temporary memory. The last arguments pointers now refer to the objects 2.8 and 4.1, and the objects 1.5 and 2.6 are no longer referenced.

Garbage Collection. From time to time the HP 48 will “hesitate” during an operation. This hesitation is usually caused by the removal of objects in temporary memory which are no longer being used. Objects which are no longer referenced continue to accumulate in temporary memory until memory has been filled. When memory is full, the calculator scans the objects in temporary memory, deleting those without references to them. This process, known as “garbage collection”, is similar in concept to garbage collection in LISP.

A large number of pointers on the stack that point to temporary memory can slow down the garbage collection process to an uncomfortable degree. This occurs when there are a large number of objects on the stack, or an object has been extracted from a large list. A worst case scenario occurs when a list that has been stored in a local variable has been broken out onto the stack using OBJ→. In this case, the time required for garbage collection increases roughly with the square of the number of objects that were in the list. List operations can be optimized by storing the lists in global variables, effectively moving the operations from temporary memory to user memory.

The MEM command returns the amount of available memory, forcing an initial garbage collection to return an accurate result. It may be helpful to insert the sequence MEM DROP to force garbage collection prior to speed-sensitive program sequences.

The NEWOB Command. The command NEWOB may be used to create a new copy of an object in temporary memory, whose only reference is on the data stack. In general, the system will perform an automatic NEWOB where it makes sense. For instance, if you recall the contents of a variable to the stack and press **[EDIT]**, the object will be copied to temporary memory before editing begins.

There are two uses for NEWOB:

- NEWOB “frees” an object that was extracted from a list. Consider the following program:

```
⌘ { "AB" "CD" "EF" } 2 GET ⌘
```

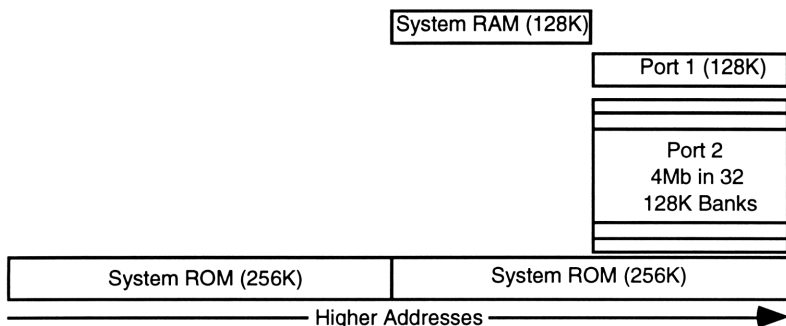
Level 1 of the data stack contains a pointer into the list, which still resides in temporary memory. Executing NEWOB now would create the unique object “CD” in temporary memory, and release the list for garbage collection. Note: set the Last Arguments flag (–55) to prevent the list from being referenced as a last argument.

- Recalling an object to the stack simply returns a pointer to the data stack. To purge a backup object from a port while retaining a copy in temporary memory, recall the object and execute NEWOB. Then the original object may be purged because there are no references to it.

Configuring RAM Cards

Initial Configurations. Before a plug-in RAM card is used, some consideration should be given to its intended use. RAM cards may be configured two ways:

- *Independent* RAM may be thought of as an “electronic disk”, which may be removed from the calculator. Individual objects or entire directories may be placed in independent RAM (see *Backup Objects* for more details). This configuration is most suitable for backing up data, “hiding” data from the HOME directory, or exchanging data with another calculator.

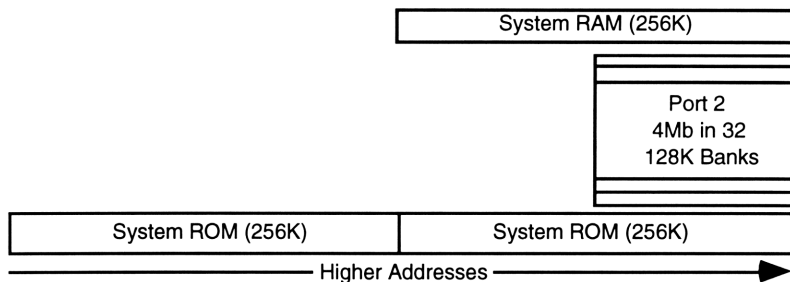


HP 48GX Memory Organization (Port 1 Independent)

Independent RAM cards in port 2 may contain up to 32 banks of 128K bytes, each of which becomes a logical *port*. When installing a large RAM card into port 2, the command PINIT is useful for initializing all the banks at once.

NOTE: Some objects, such as user programs which use new HP 48G/GX commands will not work properly when read into a HP 48S/SX.

- *Merged* RAM extends the built-in RAM, creating more room for variables and directories, port 0, temporary objects, or graphics display area. In the HP 48SX, a card used for merged RAM may be installed in either port 1 or port 2. In the HP 48GX, merged RAM can only be installed in port 1.



HP 48GX Memory Organization (Port 1 Merged)

Merged RAM may not be removed from the calculator unless the FREE1 command is used to free it. To free a card, make sure there is enough available memory to hold all your variables (including the contents of port 0), enter a blank list in level 2 and execute FREE1.

Changing Configurations. A merged RAM card may also be “converted” to an independent RAM card containing objects that were in port 0. To do this, enter a list containing the objects to transfer to independent RAM and execute FREE1.

The reverse operation is also possible. An independent RAM card may be converted into merged RAM with the MERGE1 command. Any objects that were in the card will appear in port 0.

Understanding Port 0. Port 0 is a portion of built-in memory (which may include merged RAM) which behaves in the same manner as an independent RAM card (except that it is not removable). Port 0 may contain either library or backup objects. The amount of memory devoted to port 0 changes as objects are stored in it or purged from it.

Graphics

The HP 48 display is a 131x64 pixel LCD which may present either the stack display or *PICT*, a portion of memory set aside for graphic displays. To switch the LCD between the stack display and *PICT*, use the commands *PICTURE*, *PVIEW* (to display *PICT*), and *TEXT* (to show the stack display).

Graphics objects (often known as *grobs*) may be placed into either *PICT* or the stack display, however the latter operation requires commands described in *System Programming*.

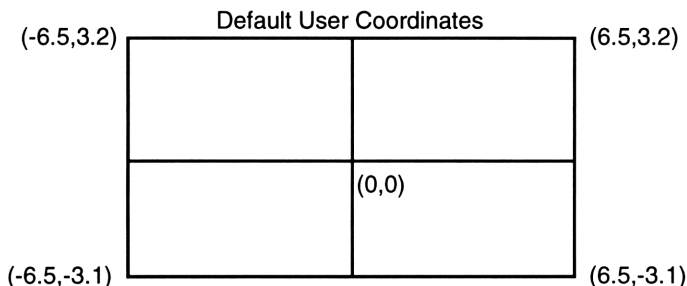
The size of *PICT* must be at least 131x64 pixels (1098 bytes), and no wider than 2048 pixels. *PICT* can be enlarged using the *PDIM* command. *PICT* may also be removed from memory with the *PURGE* command to make space available for memory-intensive applications that do not need the graphics display. If *PICT* does not exist when a command refers to it, a default size *PICT* will be created before the command is executed.

Graphics Coordinates

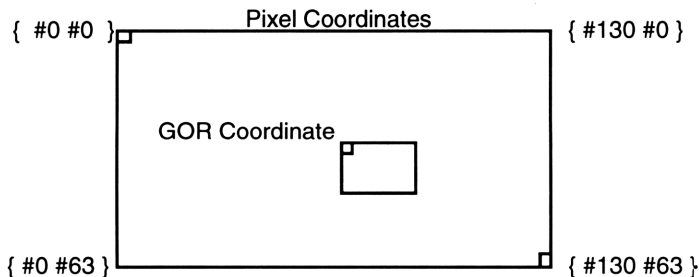
Two systems of coordinates may be used to manipulate *PICT* and graphics objects: *user-unit coordinates* and *pixel coordinates*. The reserved variable *PPAR* (which may reside in any directory) stores scaling information and parameters which pertain to plotting. The full definition of *PPAR* is given later in this chapter.

The 3D plot types use an abstract space called a *view volume* to visualize functions of two variables. The view volume is described later along with the reserved variable *VPAR*, which stores the parameters of the view volume.

User Unit Coordinates. User units, represented as complex numbers, are typically used to define the boundaries of plots. User unit scaling information is stored in the reserved variable *PPAR*. The first two entries in *PPAR* store the coordinates of the lower-left corner and upper-right corner of *PICT*. The default plot boundaries are $(-6.5, -3.1)$ and $(6.5, 3.2)$.



Pixel Coordinates. Pixel coordinates are represented by a list containing two binary integers, $\{ \#col \#row \}$. Graphics objects on the stack may only be described with pixel coordinates. The upper-left pixel of *PICT* or any graphics object is represented by $\{ \#0 \#0 \}$.



Related Commands: The commands $C \rightarrow PX$ and $PX \rightarrow C$ convert between user-unit and pixel coordinates based on the dimensions in *PPAR*. The *PDIM* command changes the size of *PICT*. *PMIN* and *PMAX* set the coordinates of the plot in user units, and *SCALE* specifies the x and y scale in units per 10 pixels. Other commands that affect scaling are *AUTO*, *AXES*, *DEPND*, *INDEP*, **H*, and **W*.

Graphics Operations

The commands GOR, GXOR, and REPL may be used to superimpose one graphics object onto another. GOR performs a logical OR for each pixel. GXOR performs an exclusive OR (useful for cursors or animation applications). REPL replaces the destination grob's pixels with those from the new grob.

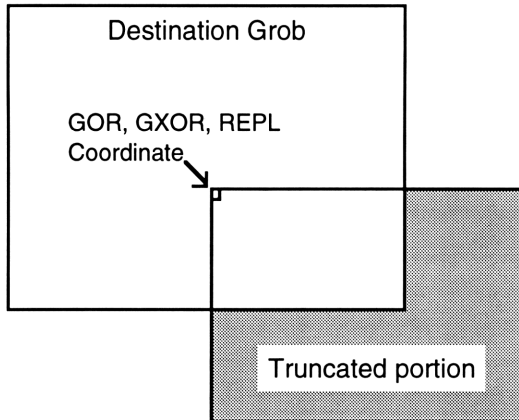
The syntax for these commands is:

Level 3	Level 2	Level 1	→	Level 1
grob ₁	{ #x #y }	grob ₂	→	grob ₃
grob ₁	(x, y)	grob ₂	→	grob ₃
<i>PICT</i>	{ #x #y }	grob	→	
<i>PICT</i>	(x, y)	grob	→	

The level 3 argument is the destination (or target) grob. The upper left corner of grob₂ will be positioned on grob₁ at the location specified in level 2. If the coordinate is specified in user-units the current scaling values in the variable *PPAR* will be used. If no copy of *PPAR* is found in the current directory, a new *PPAR* will be created with the default settings (see *PPAR*).

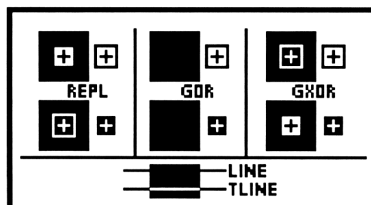
If the destination grob is anything other than *PICT*, the new grob will be returned to the stack. If the destination is *PICT*, no copy is returned.

Truncation. If grob₂ extends beyond the boundary of the destination grob it will be truncated, leaving the size of the destination grob unchanged.



Drawing Lines. The commands `LINE` and `TLINE` are available for drawing lines in *PICT*. `LINE` sets the pixels in a line between two given coordinates; `TLINE` toggles the state of the pixels in a line between two coordinates.

Example: The program `GREX` illustrates the differences between `GOR`, `GXOR`, and `REPL`, as well as between `LINE` and `TLINE`. The program begins by defining a grob showing a small cross within a box. For each command `REPL`, `GOR`, and `GXOR`, the cross is drawn on black pixels and white pixels, then the inverted cross is drawn on black and white pixels below. Then the commands `LINE` and `TLINE` are used to draw two lines through a box.



«

GROB 9 9 FF10101011101110D710111011101010FF10

PICT RCL → cross pictsav

«

```

PICT PURGE ( #0 #0 ) PVIEW
#19d #19d BLANK NEG
PICT ( #7d #1d ) 3 PICK REPL
PICT ( #49d #1d ) 3 PICK REPL
PICT ( #93d #1d ) 3 PICK REPL
PICT ( #7d #27d ) 3 PICK REPL
PICT ( #49d #27d ) 3 PICK REPL
PICT ( #93d #27d ) 3 ROLL REPL
PICT ( #12d #6d ) cross REPL
PICT ( #12d #32d ) cross NEG REPL
PICT ( #18d #21d ) "REPL" 1 →GROB REPL
PICT ( #28d #6d ) cross REPL
PICT ( #28d #32d ) cross NEG REPL
PICT ( #54d #6d ) cross GOR
PICT ( #54d #32d ) cross NEG GOR
PICT ( #61d #21d ) "GOR" 1 →GROB REPL
PICT ( #70d #6d ) cross GOR
PICT ( #70d #32d ) cross NEG GOR
PICT ( #98d #6d ) cross GXOR
PICT ( #98d #32d ) cross NEG GXOR
PICT ( #103d #21d ) "GXOR" 1 →GROB REPL
PICT ( #114d #6d ) cross GXOR
PICT ( #114d #32d ) cross NEG GXOR
PICT ( #49d #51d ) DUP2
( #67d #63d ) SUB NEG REPL
( #39d #53d ) ( #77d #53d ) LINE
( #39d #60d ) ( #77d #60d ) TLINE
PICT ( #79d #51d ) "LINE" 1 →GROB REPL
PICT ( #79d #58d ) "TLINE" 1 →GROB REPL
( #0d #48d ) ( #130d #48d ) LINE
( #43d #0d ) ( #43d #48d ) LINE
( #85d #0d ) ( #85d #48d ) LINE
( ) PVIEW pictsav PICT STO

```

»

»

Converting Text to Graphics

The command \rightarrow GROB is used to create a graphic representation of any object. \rightarrow GROB takes an object in level 2 and a font size expressed as a real number in level 1. The font number may be 0, 1, 2, or 3. There are three font sizes in the HP 48:

Number	Description
0	5x9 (fixed) or EquationWriter
1	3x5 (proportional spacing)
2	5x7 (fixed spacing)
3	5x9 (fixed spacing)

The small (3x5) font makes no distinction between lowercase and uppercase characters. This is the font used to build menu labels. Note that the small font is proportional. The character “I” is the narrowest, and characters like “M” are wider than most.

The large font (5x9) is the font used for the stack display, and shows the most detail for special characters.

Font size 0 is a special case. Except for algebraic and unit objects, objects are expressed with the large (5x9) font. Algebraic and unit objects are expressed in their form as seen in the EquationWriter, and are displayed in a grob that has a minimum size of 131x56.

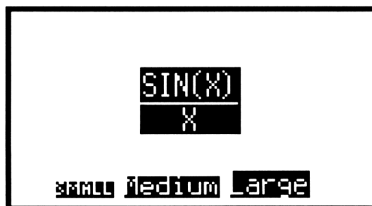
Example: The program TXTEX displays the words “Small”, “Medium”, and “Large” in the display, then displays the equation 'SIN(X)/X' in EquationWriter form. To illustrate the boundaries of the text, the graphics objects are inverted with the NEG command. The EquationWriter example, returned as a 131x56 grob, is reduced in size with the SUB command.

TXTEX 359 Bytes Checksum #96D9h

```

«
  PICT RCL →pictsav           Preserve original PICT
  «
    PICT PURGE                Purge original PICT
    PICT ( #14d #58d )
    "Small" 1 →GROB NEG GOR   Draw small font example
    PICT ( #40d #56d )
    "Medium" 2 →GROB NEG GOR Medium font example
    PICT ( #80d #54d )
    "Large" 3 →GROB NEG GOR  Draw large font example
    PICT ( #46d #15d )
    'SIN(X)/X' 0 →GROB NEG   EquationWriter example
    ( #0d #13d )
    ( #37d #37d )
    SUB                       Extract corner with equation
    GOR                       Place into PICT
    ( ) PVIEW                 Show PICT, wait for CANCEL
    pictsav PICT STO         Restore original PICT
  »
»

```



Animation Sequences

To display a sequence of graphics objects, use the sequence

```
« PICT { #0 #0 } grob REPL »
```

rather than the sequence

```
« grob PICT STO ».
```

This will be faster and will avoid flickering in the display.

The ANIMATE command is helpful for rapidly displaying a series of grobs in *PICT*. ANIMATE takes a series of grobs and either a number specifying the number of grobs to use or a list:

$$\begin{array}{l} \text{grob}_1 \dots \text{grob}_n \ n \rightarrow \\ \text{grob}_1 \dots \text{grob}_n \ \{ n \ \{ \#x \#y \} \text{ delay repeat} \} \rightarrow \end{array}$$

The list parameter must contain exactly the following four objects:

- | | |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n | The number of grobs on the stack |
| { #x #y } | The location in <i>PICT</i> (pixel coordinates) at which the upper-left corner of the grobs will be placed. |
| delay | The delay time (in seconds) between each grob. A zero delay time is fastest. |
| repeat | The number of times to cycle through the sequence of grobs. If the repeat count is zero, the program will cycle through the grobs 1048575 times or until the CANCEL key is pressed. |

ANIMATE may be interrupted by pressing **CANCEL** at any time.

Example. A series of four grobs can be used to simulate a hand moving around a clock face.

ANIM 192.5 Bytes Checksum #D062h

«

GROB 9 9 83006D00290011101110101028006C008300

GROB 9 9 83006C00280010101F10101028006C008300

GROB 9 9 83006C00280010101110111029006D008300

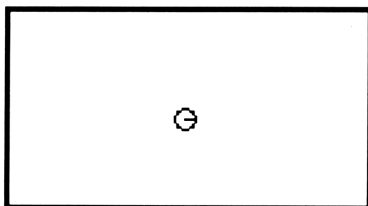
GROB 9 9 83006C0028001010F110101028006C008300

{ 4 { #61d #28d } .1 5 }

ANIMATE

5 DROPN

»



Stack View Program

The stack-view program STKV displays up to ten levels of the stack simultaneously. The display mode, plot parameters, stack values, and graphics picture are preserved. The system remains halted until **CANCEL** is pressed, then the program restores PPAR and PICT.

STKV 377.5 Bytes Checksum #6C64h

« IF DEPTH THEN	<i>Make sure stack is not empty</i>
PICT RCL PPAR	
→ pictsav ppar	<i>Preserve PICT & PPAR</i>
« PICT PURGE	<i>Purge original PICT</i>
1 32 XRNG 1 64 YRNG	<i>Set new X and Y ranges for stack</i>
1 DEPTH 1 - 10 MIN DUP	<i>Determine current stack height</i>
IF 8 >	<i>If greater than 8, text row height</i>
THEN 6 1	<i>is 6 and text size is 1</i>
ELSE 8 2	<i>Otherwise, text row height is 8</i>
END → rowht tsize	<i>and text size is 2</i>
« FOR i PICT 1 i rowht	<i>Loop for the no. of stack levels</i>
* R→C RCLF STD i	<i>Use STD display mode to build</i>
" : " + SWAP STOF	<i>stack level identifier</i>
i 3 + PICK →STR +	<i>Add stack value to identifier,</i>
tsize →GROB GOR	<i>and add to picture</i>
NEXT	<i>End loop</i>
{ } PVIEW	<i>Show PICT, wait for CANCEL</i>
'PPAR' PURGE ppar	<i>Purge new PPAR</i>
IF 'PPAR' SAME NOT	<i>Did PPAR exist before?</i>
THEN ppar 'PPAR' STO	<i>Yes, store old value</i>
END pictsav PICT STO	<i>Restore original PICT</i>
»	
»	
END	
»	

```
10: 247
9: '6.5 M/S'
8: STRING
7: (1.1,2.2)
6: TAG: 21.54
5: GROB 131 64 00000000000000000000
4: 'Y=2XX^2+3XX-4'
3: # 25ERH
2: [ 1 2 3 4 5 ]
1: { 1 "AB" }
```

Graphics Object Structure

A graphics object is structured as follows:

`<header><length><height><width><data...>`

- | | |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| header | This is a five nibble (1/2 byte) field that distinguishes a graphics object from any other object type, and has a fixed value of #02B1Eh. |
| length | This field is a five-nibble quantity that contains the distance in nibbles from the start of the length field to the nibble past the end of the object. This length is #Fh + the number of data nibbles. |
| height | This field is a five-nibble quantity that specifies the height of the graphics image in pixels. |
| width | This field is a five-nibble quantity that specifies the width of the graphics image in pixels. |
| data | The data nibbles begin at the upper-left corner of the graphics object and proceed left-to-right, top-to-bottom. Each row must contain an integral number of bytes, so the data may be padded with garbage bits. The bits in each nibble are written in reverse order, so the leftmost displayed pixel in a nibble is represented by the least-significant bit of the nibble. |

If you are preparing a graphics object on a personal computer, remember that the HP 48 CPU reads data from memory into registers in reverse order, so the first four fields are written backwards. For example, the header is written E1B20.

Graphics objects may be entered into the command line on the HP 48. To enter a blank graphics object, type **GR0B** *width height*, where *width* and *height* specify the size in pixels.

Examples. To enter a graphics object which represents “G” in the small font, type `GROB 4 5 E010D090E0`.

	●	●	●
●			
●		●	●
●			●
	●	●	●

On a personal computer, the graphics object looks like this:

E1B20	B1000	50000	40000	E010D090E0
<i>header</i>	<i>length</i>	<i>width</i>	<i>height</i>	<i>data</i>

In the second example consider a blank graphics object that is the size of the display with the “G” from above in the upper-left corner. The graphics object looks like this on a personal computer:

[illegible]

The fields for the example of the previous page are derived as follows:

- The display width is 131 columns = 83h pixels, or 17 bytes or 34 nibbles.
- The display height is 64 rows = 40h pixels.
- The data length is bytes-per-row x rows = 2176 nibbles. The length field is calculated as $2176 + 15 = 2191d = 88Fh$.

PPAR

The reserved variable *PPAR* (which may exist in every directory) contains scaling information and plot specifications.

PPAR		
{ (X _{min} , Y _{min}) (X _{max} , Y _{max}) indep resolution (X _{axis} , Y _{axis}) ptype depend }		
Parameter	Description	Default
(X _{min} , Y _{min})	Lower-left pixel coordinates	(-6.5, -3.1)
(X _{max} , Y _{max})	Upper-right pixel coordinates	(6.5, 3.2)
indep	Independent variable for horizontal axis	X
resolution	Real positive integer for user-unit point spacing, or binary integer for pixel spacing (0= every column).	0
(X _{axis} , Y _{axis})	Axes intersection coordinates, tick specification, and axes labels.	(0,0)
ptype	Plot type: FUNCTION, CONIC, BAR, POLAR, DIFFEQ, PARAMETRIC, HISTOGRAM, SCATTER, TRUTH, SLOPEFIELD, WIREFRAME, YSLICE, PCONTOUR, GRIDMAP, or PARSURFACE.	FUNCTION
depend	Dependent variable	Y

The lower-left and upper-right pixel coordinates are used to initialize VPAR parameters if VPAR hasn't been specified for 3D plots.

INDEP and DEPEND. The parameters *indep* and *depend* specify the independent and dependent variables, and can specify a *plotting range*, which is used to control the range of a plot for plot types such as polar and parametric. The full definition for these parameters is:

```
indep:    { name x_start x_end }
depend:   { name y_start y_end }
```

For BAR plots, x_{start} specifies the horizontal location of the first bar.

If the plot type is DIFFEQ, *indep* and *depend* are used as follows:

```
indep:    { name t_0 t_f }           Default: { 'T' 0 x_max }
depend:   { name y_0 ErrorTolerance } Default: { 'Y' 0 0 }
```

RES. The *resolution* parameter specifies the interval between plotted points of the independent variable in user (real number) or pixel (binary integer) units. The default value is 0, which specifies one point every column. Some plot types use *resolution* differently:

BAR	Specifies the width of each bar
DIFFEQ	Specifies the maximum interval (unlimited if 0). The initial interval is the smaller of $\ t_f - t_0\ $ and <i>res</i> , or $\ t_f - t_0\ /131$ if <i>resolution</i> =0.
HISTOGRAM	Specifies the bin width in pixels. The default bin size is 1/13 of the difference between the minimum and maximum data values.
PARAMETRIC	The default value of 0 specifies an interval that is 1/130 of the difference between x_{start} and x_{end} .
POLAR	The default value of 0 specifies 2° , 2 grads, or $\pi/90$ radians.

The plot types PARSURFACE, PCONTOUR, SCATTER, SLOPEFIELD, and WIREFRAME do not use *resolution*.

AXES. The *axes* parameter can specify the intersection of the plot axes, the spacing of tick marks, and axes labels. The tick mark specification and the labels are optional. The full definition for the axes parameter is:

{ intersection tick “x-label” “y-label” }

where:

intersection The axes intersection is specified as a complex number.

tick The *tick* parameter may contain a tick spacing parameter expressed in either user-units or pixels. User unit spacing is expressed as a real number specifying the spacing for both the x and y axes, or a list containing two real numbers which specify the x and y spacing. Pixel spacing is expressed as a binary integer number of tick marks for both axes, or a list of two binary integers which specify the x and y spacing.

Examples:

5	5 user unit spacing on each axis.
{ 1 10 }	1 unit on the x-axis, and 10 units on the y-axis.
#5d	5 pixel spacing on each axis.
{ #5 #10 }	5 pixels on the x-axis, and 10 pixels on the y-axis.

“x-label” Horizontal axis label.

“y-label” Vertical axis label.

If the plot type is DIFFEQ, the axes labels are used as follows:

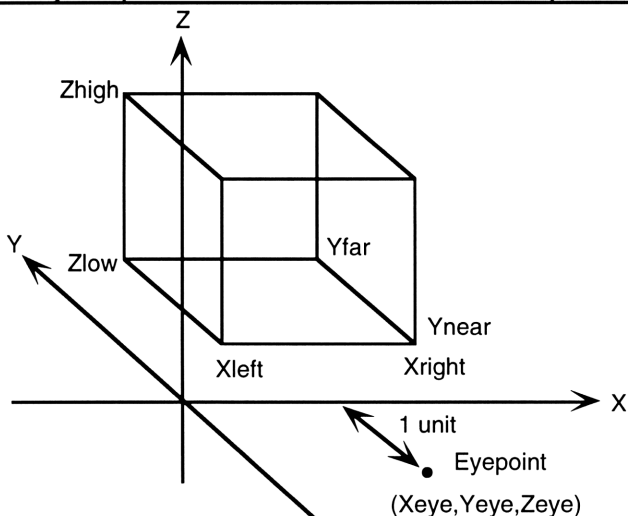
“x-label”: Index for which independent variable, vector-valued solution to plot on the horizontal axis “1”, “2”, etc.

“y-label”: Index for which dependent variable, vector-valued solution to plot on the vertical axis “1”, “2”, etc.

VPAR

The view volume is a region in abstract 3-dimensional space used by the 3D plot types. The reserved variable *VPAR* stores the dimensions of the view volume, the location of the eye point, the input ranges for gridmap and parsurface plots, and the number of plotting steps. *VPAR* can reside in any directory, like *EQ* and *PPAR*.

VPAR		
{ X_{left} X_{right} Y_{near} Y_{far} Z_{low} Z_{high} X_{min} X_{max} Y_{min} Y_{max} X_{eye} Y_{eye} Z_{eye} X_{step} Y_{step} }		
Parameter	Description	Default
X_{left} X_{right}	Width of view volume	-1 1
Y_{far} Y_{near}	Depth of view volume	-1 1
Z_{low} Z_{high}	Height of view volume	-1 1
X_{min} X_{max}	X-input range for GRIDMAP and PARSURFACE plots	-1 1
Y_{min} Y_{max}	Y-input range for GRIDMAP and PARSURFACE plots	-1 1
X_{eye} Y_{eye} Z_{eye}	Coordinates of the eyepoint	0 -3 0
X_{step} Y_{step}	Number of plotting steps	10 8



Programming

The HP 48 provides several avenues for customization: user-defined keys, the CST menu, equations supplied to the solver, user-defined functions, and *programs*, which offer the most flexibility for customization.

In the simplest form, a program is an object consisting of a collection of commands or functions enclosed by program delimiters (« »). A simple example returns the circumference of a circle given its radius in level 1:

« 2 * π * →NUM »

When stored in a variable, this program can be executed simply by pressing its menu key in the VAR menu or including its name in another program.

Program Execution

When a program is executed, the objects that make up the program have actions that are specific to their type. Data class objects (such as numbers, arrays, strings, algebraic objects, or lists) or programs are placed on the stack. Quoted global or local variable names are placed on the stack. If a local variable name is unquoted, its contents are placed on the stack. An unquoted global variable name will have behavior dependent on the contents of the variable:

Contents	Action
Program	Executes the program
Name	Evaluates the name
Directory	Makes <i>directory</i> current
Any other object	Puts contents on the stack

To illustrate the evaluation of variable names, store the number 8 in variable C, the name 'C' in variable B, and the name 'B' in variable A. The program « A » will now return the value 8.

Single-Stepping Programs

The ERROR menu (**PRG** **NXT** ERROR) provides several tools for single-stepping through a program. There are two ways to halt execution within a program:

- Place the name of the program on the stack and execute **DEBUG**. This will begin execution of the program, but the program will be suspended at the first step.
- Insert the **HALT** command in your program where you'd like to suspend execution. When executed, the program will be suspended and ready to execute the object after **HALT**.

Once the program has been suspended, the following operations are available:

- | | |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| SST | Executes the next object in the program. If the next object is a program, SST executes the whole program as one step, as opposed to SST+ . |
| SST+ | If the next object is a named program, single-steps <i>into</i> the program, otherwise just executes the object. |
| NEXT | Displays the next one or two objects in the program. |
| ⏮ CONT | Resumes program execution. The program will continue until another HALT command is encountered or the program ends. |
| KILL | Cancels all suspended programs. |

Local Variables

Programs which use a large number of variables and/or intermediate results may use *local variables* to provide intermediate storage off the stack.

The formal syntax for programs using local variables is:

⌘ → *local-names defining-procedure* ⌘

If the values for local variables are established at the start of the program, the syntax looks like this:

⌘ *establish-values* → *local-names defining-procedure* ⌘

The defining procedure may be either an algebraic expression or a program:

⌘ → *local-names* 'algebraic' ⌘

⌘ → *local-names* ⌘ *program* ⌘

Local variables exist in a local (temporary) *environment* during execution of the defining procedure and take precedence over global variables of the same name when evaluated. Temporary environments may be nested (programs within programs). In this case, evaluation of a local variable name returns the contents of the most recently defined local variable of that name.

When stack objects are first stored in local variables by the → operator, the object in level 1 is stored in the last name in the series of local variables. By convention, local variables are written in lowercase.

There are a number of advantages that come with the use of local variables:

- Conflicts with global variables are avoided.
- Storing to a local variable is quite fast, because temporary environments are typically quite small, and avoid the overhead of moving all the data in global variables. Depending on the number of global variables in the current path, fewer local variables may be searched, reducing the time required for programs to store & recall objects.

- Temporary environments are automatically removed at the end of the defining procedure, so the defining procedure does not have to spend time and extra code “cleaning up” after itself.
- Programs are often easier to write (and later on, to read) when intermediate values are stored in named local variables than programs which keep everything on the stack. (Imagine remembering which value resides in stack level 22 two years after you write a program.)

Example: To illustrate the action of \rightarrow as it stores objects on the stack into local variables, consider the program LCLV:

LCLV 78.5 Bytes Checksum #7AA3h

```

«  $\rightarrow$  a b c
«
  a "a"  $\rightarrow$  TAG
  b "b"  $\rightarrow$  TAG
  c "c"  $\rightarrow$  TAG
»
»
»

```

The program LCLV tags the objects stored in local variables with the names of the local variables. If you have the number 3 in level 3, 2 in level 2, and 1 in level 1, the results from LCLV are:

3 2 1 \rightarrow a:3 b:2 c:1

Example: Suppose the stack contains the same values as above (3 in level 3, 2 in level 2, and 1 in level 1). The following programs produce the same result (17) by first assigning the values to local variables x , y , and z :

```

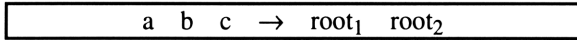
«  $\rightarrow$  x y z '(x*y+z)*2+x' »
«  $\rightarrow$  x y z
« x y * z + 2 * x + »
»

```

Example: Consider two approaches to finding the roots of a quadratic equation $x=ax^2+bx+c$. We'll use the quadratic formula:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The stack diagram for these program examples will be:



Each example program will bind the values for a, b, and c into local variables a, b, and c.

The first program, QRT1, places the intermediate result $\sqrt{b^2-4ac}$ on the stack:

QRT1 116 Bytes Checksum #6DE1h

<pre> « → a b c « b SQ a c * 4 * - √ b NEG OVER + a 2 * / b NEG ROT - a 2 * / » » </pre>	<p><i>Create local variables</i></p> <p><i>Calculate $\sqrt{b^2-4ac}$</i></p> <p><i>Calculate first root</i></p> <p><i>Calculate second root</i></p>
------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------

The second program, QRT2, uses two algebraic expressions. This version is somewhat longer, but easier to read:

QRT2 159.5 Bytes Checksum #6C6Dh

```
« → a b c
«
'(-b+√(b^2-4*a*c))/(2*a)' →NUM
'(-b-√(b^2-4*a*c))/(2*a)' →NUM
»
»
```

Programs as Subroutines

The HP 48 has no formal notion of a *subroutine*, as defined in other languages such as BASIC. A subroutine in the HP 48 is simply another program which follows the same basic rules as all other programs. The moniker “subroutine” or “subprogram” may be applied to any HP 48 program for the sake of clarity or convention. Subroutines may be stored in global variables to simplify the structure of a program. If you’re writing an application that will be distributed to other people, it’s best to avoid excessive use of global variable usage for subroutines and intermediate results.

Two techniques are available for minimizing the use of global variables. The first is to store subroutines in local variables, the second is to use local variable compilation to reduce the use of global variables for intermediate results. Some programs in the chapter *Example Programs* use these techniques.

Subroutines in Local Variables. If you don’t want to store subroutines in global variables, you can store them in local variables.

When a local variable name is evaluated, it *only recalls* the contents of the variable. This is similar to evaluating global names that contain data objects. Consequently, if the local variable contains a program, it can only be executed by an explicit EVAL. To improve legibility, local variables containing subroutines are often named in initial caps (like *Subr*).

Example: To illustrate the use of subroutines in local variables, consider a modification of the program QRT1, where the sequence

2 * /

is placed in a subroutine:




QRT3 148.5 Bytes Checksum #89F1h

```

«
« 2 * / »           Place subroutine on stack
→ a b c Subr       Create local variables
«
  b SQ a c * 4 * - √   Calculate  $\sqrt{b^2-4ac}$ 
  b NEG OVER + a Subr EVAL Calculate first root
  b NEG ROT - a Subr EVAL Calculate second root
»
»
»

```

Compiling Local Variable Names. Ordinarily, local variables (and hence local variable name objects) exist only within the context of the defining object. This yields a problem for sharing data between two programs stored in global variables or a subroutine stored in a local variable which needs to access local variables from the defining procedure. Until the arrival of the HP 48G/GX models, the only way to create a program stored in a global variable that uses another program's local variables was to HALT the program containing the defining procedure, then enter and store the new program. In practice this is not satisfactory, because you cannot edit the program without once again setting up the temporary environment.

In the HP 48G/GX, variable names which are prefixed with the character \leftarrow (  ) are compiled as local variable name objects, and thus can access temporary environments when evaluated, even though the temporary environment does not yet exist when the program is entered into the HP 48.

Example: Notice that both uses of the program Subr in the program QRT3 are preceded by the execution of the local name object *a*. These two instances may be combined into Subr by using \leftarrow to specify *a* as a local name object.

QRT4 147 Bytes Checksum #7720h

```

«
  «  $\leftarrow a$  2 * / »           Place subroutine on stack
  →  $\leftarrow a$  b c Subr          Create local variables
  «
    b SQ  $\leftarrow a$  c * 4 * - √   Calculate  $\sqrt{b^2-4ac}$ 
    b NEG OVER + Subr EVAL      Calculate first root
    b NEG ROT - Subr EVAL       Calculate second root
  »
»

```

Example: The subroutine can also be stored in a global variable:

SUBR 31.5 Bytes Checksum #8CC2h

```

«  $\leftarrow a$  2 * / »

```

QRT5 109 Bytes Checksum #CDD1h

```

«
  →  $\leftarrow a$  b c                Create local variables
  «
    b SQ  $\leftarrow a$  c * 4 * - √   Calculate  $\sqrt{b^2-4ac}$ 
    b NEG OVER + SUBR           Calculate first root
    b NEG ROT - SUBR            Calculate second root
  »
»

```

Here you see that the program SUBR can be entered independently of the program QRT5. One implication of this technique is that one program can be used by several other programs without the burden of sharing data through global variables.

User-Defined Functions

User-defined functions may be used to extend the function set of the HP 48. A user-defined function takes its arguments from the stack and must return exactly one result to the stack. The arguments may be either algebraic or numeric.

The syntax of a user-defined function must be exactly:

« → *local-names defining-procedure* »

User-defined functions created with the DEFINE command use an algebraic expression as the defining procedure. If the defining procedure is a program, the program must remove all arguments from the stack and return one real number.

The DEFINE command simplifies the creation of a user-defined function by converting an expression in the form

'name⟨arguments⟩=expression'

into a named program that consists of a local variable structure and an algebraic expression. A user-defined function may be stored into a local variable (see the program TREE in *Example Programs* for an illustration of this technique).

The example program UDFUI shows how an input form can be created from a user-defined function.

Example: Create a function $POLY(x)=2x^2+4x+7$. Enter the expression 'POLY(x)=2*x^2+4*x+7' and execute DEFINE. The variable POLY now contains the program:

« → x '2*x^2+4*x+7' »

If the number 8 is in level 1, executing POLY yields 167. Assuming that the variable S is undefined, POLY('S+5') yields the expression '2*(S+5)^2+4*(S+5)+7'.

Example: Create a function $PTHG(x,y)=\sqrt{x^2+y^2}$. Enter the expression 'PTHG(x,y)=√(x^2+y^2)' and execute DEFINE. The variable PTHG in the VAR menu now contains the program:

« → x y '√(x^2+y^2)' »

Looping Structures

Program loops are useful for repetitive execution of a procedure. There are two general classes of loops:

- *Definite loops* execute a *loop-clause* at least once, and execute a predefined number of iterations.
- *Indefinite loops* execute a *loop-clause* repeatedly until a *test-clause* returns a true (non-zero) result. One form of an indefinite loop may not execute at all if an initial test fails.

Definite Loops. There are two types of definite loops, both of which can have an increment of either 1 or n :

start finish FOR *index* *loop-clause* NEXT

start finish FOR *index* *loop-clause* *increment* STEP

start finish START *loop-clause* NEXT

start finish START *loop-clause* *increment* STEP

In each case the *start* and *finish* values are taken from the stack and are no longer available to the program. The *index* is a local variable that may be referenced in the loop clause just like any other local variable. The *increment* is also taken from the stack. This syntax shows it being put there explicitly by the program, but it can be calculated also.

	Increment=1	Increment= n
Index	FOR ... NEXT	FOR ... n STEP
No Index	START ... NEXT	START ... n STEP

The differences are:

- FOR loops keep their index in a local variable which is available to the loop-clause. An early exit may be taken from a FOR loop by one of the following two methods:
 - Store MAXR in the index for loops with a positive step.
 - Store -MAXR in the index for loops with a negative step.
- START loops save memory and execute faster than FOR loops for applications where access to the index is not needed.
- Loops ending with STEP may have a varying increment. When STEP is executed, the increment is added to the index. The loop will repeat under the following conditions:
 - The increment is positive and the incremented index is less than the finish value.
 - The increment is negative and the incremented index is greater than the finish value.
- Loops ending with NEXT execute faster than those ending with STEP, because the increment value is always 1.

Examples:

« 1 10 START *loop-clause* NEXT »

Executes *loop-clause* 10 times.

« 1 20 FOR \times *loop-clause* NEXT »

Executes *loop-clause* 20 times; x is the index.

« 1 10 START *loop-clause* 2 STEP »

Executes *loop-clause* 5 times.

« 1 20 FOR \times *loop-clause* 2 STEP »

Executes *loop-clause* 10 times; x is the index.

Indefinite Loops. There are two forms of indefinite loops:

- **DO** *loop-clause* **UNTIL** *test-clause* **END**

DO loops execute at least once. The placement of UNTIL is not important since the test occurs at the end, but by convention is placed between the loop and test clauses to improve legibility.

- **WHILE** *test-clause* **REPEAT** *loop-clause* **END**

WHILE loops never execute if the test-clause returns an initial false (zero) result. The placement of REPEAT is important, as it isolates the test clause, which usually executes one time more than the loop clause.

Loop Counters. The commands INCR and DECR may be used at any time to increment or decrement a real number stored in a variable.

The command INCR takes a local or global variable name, increments its contents, and returns the new value to the stack. For instance, if x contains 23, 'x' INCR stores 24 in x and returns 24 to the stack. DECR behaves the same way as INCR, but decrements the variable's contents.

Examples: The first program (46.5 bytes, checksum #9910h) *always* prints at least one carriage-right, up to the number of carriage-rights specified in level 1. The second program (51.5 bytes, checksum #449Fh) prints the number of carriage-rights specified in level 1.

```
« → x
  « DO CR UNTIL 'x' DECR NOT END »
»

« DUP → x
  « WHILE REPEAT CR x DECR END »
»
```

The program DSORT uses INCR (see *Sorting Directories*).

Conditional Structures

IF Structures. The IF structures perform a test and execute a *true-clause* if the test is true or a *false-clause* if the structure includes ELSE.

IF		IF	<i>test-clause</i>
	<i>test-clause</i>	THEN	<i>true-clause</i>
THEN		ELSE	<i>false-clause</i>
	<i>true-clause</i>	END	
END			

Example: This program (82.5 bytes, checksum #ACF0h) stores a value from the stack into variable a and returns .35*a or .45*a if a > 10.

```
« → a
  « IF 'a>10'
    THEN .45
    ELSE .35
    END
    a *
  »
»
```

IFT and IFTE. IFT and IFTE may be used as commands, taking their arguments from the stack. IFTE may also be used in an algebraic expression.

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

Level	IFT	IFTE
3:		<i>test-result</i>
2:	<i>test-result</i>	<i>true-clause</i>
1:	<i>true-clause</i>	<i>false-clause</i>

CASE Structures. The CASE ... END structure combines a series of IF ... THEN structures that ends when the first true condition has been met. A “default” clause may be placed before the END command which is executed if none of the conditions have been met.

CASE				
	<i>test-clause</i>	THEN	<i>true-clause</i>	END
	<i>test-clause</i>	THEN	<i>true-clause</i>	END
	...			
	<i>test-clause</i>	THEN	<i>true-clause</i>	END
	<i>default-clause</i>			
END				

Example: This program (127 bytes, checksum #A7F1h) accepts an object and issues an error for non-real types, then executes the procedure *Xneg* for numbers less than zero, *Xzero* for numbers equal to zero, or *Xpos* in the default case.

The type for a real number is zero, so a non-real object generates a true condition. In this case the command DOERR will issue message #202h, “Bad Argument Type”.

```

«
  → x
  « CASE
    x TYPE THEN # 202h DOERR END
    'x<0' THEN Xneg END
    'x==0' THEN Xzero END
    Xpos
  »
»

```

Error Trapping

The IFERR structure is useful for trapping anticipated errors. The *trap-clause* is executed first, and if no error is encountered (or the **CANCEL** key has not been pressed) an optional *ELSE normal-clause* is executed.

IFERR	<i>trap-clause</i>	IFERR	<i>trap-clause</i>
THEN	<i>error-clause</i>	THEN	<i>error-clause</i>
END		ELSE	<i>normal-clause</i>
		END	

If an error occurs within the trap clause, the following steps are taken:

- 1) The error is suppressed – the program will not be halted.
- 2) The keyboard input buffer is flushed.
- 3) If the Last Arguments flag (-55) is clear (default), the arguments to the command that generated the error are returned to the stack.
- 4) All three regions of the stack display are “unfrozen” (see *Displaying Results* for a description of the display regions and the FREEZE command).
- 5) The *error-clause* is executed.

Example: This program (63 bytes, checksum #29FBh) takes an object from the stack and returns the object’s size in bytes. If the object is a name and no variable of that name exists, the program returns 0.

```
«
  IFERR BYTES
  THEN IF -55 FC? THEN DROP END 0
  ELSE SWAP DROP END
»
```

Example: Trapping `CANCEL`. The IFERR structures can be used to trap the error generated by the `CANCEL` keystroke (error number 0). The program TRPCN illustrates this by embedding an infinite WHILE loop within an IFERR trap. As a counter is incremented, its value is displayed on successive lines of the display. The MOD function is used to wrap the display sequence from the bottom of the display back to the top. When `CANCEL` is pressed, the program could be anywhere in the trap clause when the branch to the error clause occurs. The error clause compares the current number of objects on the stack with the original number and drops the difference.

TRPCN 124.5 Bytes Checksum #1DD0h

```

❖ DEPTH                                Save stack depth
  → depth
❖ IFERR                                Set error trap
  0                                    Initial counter value is 0
  WHILE 1                             Loop forever
    REPEAT
      DUP DUP 7 MOD 1 +                Calculate display line number
      DISP .1 WAIT                     Display counter value, pause
      1 +                              Increment counter
    END
  THEN
    DEPTH depth - DROPN                Error clause cleans up stack
  END
❖
❖

```

Error Interpretation. The commands ERRM and ERRN return the most recent error message and error number. ERR0 clears the error number. These commands may be useful in an error clause for taking specific action for different kinds of errors.

User-Defined Errors. The command DOERR accepts either a system error number or a string. If the error number is zero, the action is equivalent to pressing `CANCEL`, and ERRM and ERRN are set to "" and 0. If a string is supplied, the string will be returned by ERRM and the error number will be set to #70000h.

Data Entry

A program may halt to obtain user input using a wide variety of techniques. These techniques have varying levels of restrictions on keyboard and stack operations, as well as varying cosmetic appeal.

The WAIT Command. When a single keystroke is required, the WAIT command is the preferred method. WAIT places the HP 48 into a light sleep mode to conserve batteries and returns the next keystroke in rc.p format. The rc.p encoding refers to the *row*, *column*, and *shift plane* location of the key that was pressed.

Shift Planes			
p	Primary Planes	p	Alpha Planes
0 or 1	Unshifted	4	Alpha
2	Left-shifted	5	Alpha left-shifted
3	Right-shifted	6	Alpha right-shifted

The WAIT command is interrupted when the **CANCEL** key is pressed. If an application *must* intercept **CANCEL**, use an IFERR trap or the System-RPL object WaitForKey described under *Keyboard Control* in the chapter *System Programming*.

The HALT Command. When HALT is executed, program execution stops and the stack is displayed. The program resumes when the command CONT is executed or the user presses **⏏**(CONT). The stack is available for use while a program is HALTed. A program that has been HALTed may be completely terminated by executing KILL.

The PROMPT Command. When PROMPT is executed the program displays a message and halts until CONT or KILL are executed. Using PROMPT is equivalent to the sequence:

```
« ... "string" 1 DISP 3 FREEZE HALT ... »
```

The stack is available for use while the program is halted.

The INPUT Command. The INPUT command suspends program execution, displays a message and default answer, and waits for user input. The INPUT command terminates and program execution resumes when **ENTER** is pressed. The parameters supplied to INPUT provide considerable control over the appearance of the display and cursor placement. The stack is *not* available in this state, but menus may be changed.

The parameters for INPUT can be specified in varying levels of detail:

```

“message” ”prompt” → ”result”
“message” { mode(s) } → ”result”
“message” { “prompt” column } → ”result”
“message” { “prompt” mode(s) } → ”result”
“message” { “prompt” column mode(s) } → ”result”
“message” { “prompt” { row column } mode(s) } → ”result”

```

In the simple case, “prompt” is the default answer, which will be returned to the stack if **ENTER** is pressed immediately.

The cursor position may be specified as a column number or as a list specifying the row and column number. A zero value for the column position specifies the end of the row. Rows are numbered beginning with 1 for the top row of the prompt text. A zero value for the row number specifies the last row.

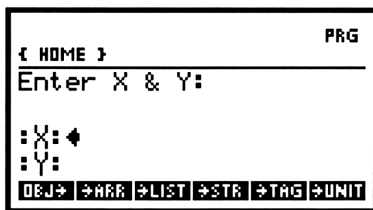
The cursor type is controlled by the value and sign of the column number (or row number if both row and number are specified). If this number is negative, the replace cursor is displayed. If this value is zero or positive, the insert cursor is displayed.

The modes may be any of the following (entered as unquoted names):

- α Locks alpha entry on for text entry
- ALG Selects Algebraic/Program-entry mode. The default entry mode is Program-entry.
- V Verifies that the result string represents one or more valid object. If the string fails this test, the error message “Invalid Syntax” is displayed and INPUT does not terminate.

INPUT Example: The following program fragment (69.5 bytes, checksum #C169h) prompts for two values using tags in the prompt string to help enter and delimit the results. The `▀` character in the prompt represents a newline character (`␣`). The cursor is positioned at the end of the first row, and V is used to ensure two valid objects are contained in the result string.

```
« "Enter X & Y:" { ":X:▀:Y:" { 1 0 } V } INPUT »
```



If you're writing a program for yourself, INPUT might be just the trick for setting up a prompt. If you're writing a program that needs to prompt for multiple values and will be distributed to a wide audience, there are three things to consider:

- 1) The display above shows the PRG TYPE menu. You may wish to display the EDIT menu before executing the INPUT command. This can be done by executing 28 MENU or 28 TMENU. A custom menu can also enhance the use of the INPUT command. There's an example of this on the next page.
- 2) The action of the **CANCEL** key should be considered. Where there is data in the command line, the first press of **CANCEL** acts to clear the command line. When there is no data in the command line **CANCEL** terminates the program.
- 3) Remember that regardless of the contents of the prompt message, the use of the V option, or the use of a custom menu, the user can still enter any number of objects of arbitrary type. It's a fair amount of work to create a truly bullet-proof program that uses INPUT to return *exactly* the number of objects that you want and that those objects are of the right type. The INFORM command (described later) might be a better tool in some cases.

Data Entry With Custom Menus. A custom menu can be used to provide significant additional flexibility when used in conjunction with the INPUT, PROMPT, or WAIT commands:

- INPUT: A custom menu provides typing aids.
- PROMPT: A custom menu can provide execution objects which optionally include CONT to resume program execution.
- WAIT: A custom menu can provide menu key labels for single keystroke responses, such as menu keys YES or NO.

Example: INPUT with Custom Menu. The following program fragment (102 bytes, checksum #9067h) accepts a string while providing a menu of common responses. The MENU command at the end of the program restores the previous menu.

```
«
{ "RED" "ORG" "YEL" "GRN" "BLU" "WHT" }
TMENU "Enter a color code:" "" INPUT 0 MENU
»
```

Example: PROMPT with Custom Menu. The program PRMCNT displays a simple menu which stores zeros or accumulates numbers into variables A and B. When DONE is pressed the CONT command continues the program, which then displays the sums of A and B.

PRMCNT 281 Bytes Checksum #1261d

```
«
0 'A' STO 0 'B' STO
{ { "CLRA" « 0 'A' STO » }
  { "CLRB" « 0 'B' STO » }
  { "A" « 'A' STO+ » }
  { "B" « 'B' STO+ » }
  ""
  { "DONE" CONT }
} TMENU
"Key values into A & B" PROMPT
A "A" →TAG B "B" →TAG 0 MENU
»
```

Example: WAIT with Custom Menu. The program WCM displays a menu, waits for a `CANCEL` or `OK` menu key response, beeps on invalid keys, and returns 0 if `CANCEL` was pressed or 1 if `OK` was pressed.

WCM 164 Bytes Checksum #21E3h

```

«
{ "" "" "" "" "<AN<L" "OK" }
TMENU
DO -1 WAIT                Wait for a key.
UNTIL
    { 15.1 16.1 }
    SWAP POS              POS returns 0 for invalid key.
    IF DUP                If the key was in the list...
    THEN 1 -              Subtract for 0 or 1 result
                        and signal end of loop.
    ELSE 1420 .074 BEEP   Otherwise beep and let the 0 result
    END                  from POS continue the DO loop.
END
0 MENU                  Restore the previous menu.
»

```

Note that the left parenthesis “(“ is used instead of the character “C” for the menu label.

The INFORM Command. The INFORM command provides one of the most powerful data entry options in the HP 48. The use of input forms in the HP 48 allows you to enter data into labeled fields, then press `OK` to continue the program. The behavior of these input forms is much like the input forms used for applications like Plot. The parameters to INFORM are supplied in five stack levels:

```

“Title” {Field_Specifiers} Format_Options {Reset_Values} {Initial_Values}
→
{Final_Values} 1 or 0

```

If the the input form is terminated by pressing `OK` or `ENTER` the final values are returned in level 2 and the number 1 is returned in level 1. If the input form is terminated by pressing `CANCEL` or `CANCEL` the number 0 is returned to level 1.

The parameters supplied to **INFORM** are specified as follows:

“Title” The input form title specified as a string.

{ Field_Specifiers } The field specifiers contain a label for each field, an optional help string and optional data type specifiers. A data type specifier is a real number corresponding to the type of object allowed (see *Object Types*). Each field specifier can take the following forms:

```
“LABEL”
{ “LABEL” “HELP” }
{ “LABEL” “HELP” TYPE1 ... TYPEn }
{ }
```

If the field specifier is an empty list, the field to its left will expand to span two columns. Empty list field specifiers do not require corresponding values in the reset and initial value lists supplied in stack levels 2 and 1.

Format_Options The format options specify the number of columns in which the fields will be displayed, and an optional tab width (the spacing between the first character of the field label and the data):

```
{ } Default: 1 column, tab width 3 columns
Columns
{ Columns Tab_width }
```

{ Reset_Values } The reset values are the values to be placed in the fields when **RESET** is pressed. The initial value are the values placed in the fields when **INFORM** is first executed. These lists may be empty or may contain objects and/or the placeholder command **NOVAL** to indicate unspecified fields:

{ Initial_Values }

```
{ }
{ field-obj1 NOVAL2 obj3 ... }
```

The arrangement of fields in the display is restricted to four rows of one or more columns of fields. The portion of the display above the menu keys is reserved for the help message and the command line. A wide variety of input forms can be designed by using various combinations of formatting options. To put the most information possible in the display:

- Use small field names where possible.
- Keep the tab spacing to a minimum size (0 or 1 is best). In cases where you have just one column, a larger tab width may be more cosmetically appealing.
- Combine small fields onto one line.

Example. By specifying three columns and a tab width of 1, you can place twelve fields in the display (as long as the field labels are small).

INF1 194.5 Bytes Checksum #A2F7h

```
«
"TITLE"
{
  "A:" "B:" "C:" "D:" "E:" "F:"
  "G:" "H:" "I:" "J:" "K:" "L:"
}
{ 3 1 }
{ 1 2 3 4 5 6 7 8 9 10 11 12 }
DUP INFORM
»
```

TITLE		
A: 1	B: 2	C: 3
D: 4	E: 5	F: 6
G: 7	H: 8	I: 9
J: 10	K: 11	L: 12
<div> <div>EDIT</div> <div></div> <div></div> <div></div> <div>CANCEL</div> <div>OK</div> </div>		

Example. The program INF2 displays an input form for a hypothetical phone list application. The use of empty list field specifiers provides for wide fields for the name and phone number. The phone number field does not really need to be wide, but doing this moves the email field to the last line.

INF2 349.5 Bytes Checksum #A42Ah

```
«
"DEPARTMENT LIST"
(
  { "NAME:" "ENTER NAME" 2 }
  { }
  { "EMP#:" "ENTER EMPLOYEE NUMBER" 0 }
  { "BLDG:" "ENTER BUILDING NUMBER" 0 }
  { "PHONE#:" "ENTER PHONE EXTENSION" 0 }
  { }
  { "EMAIL:" "ENTER E-MAIL ADDRESS" 2 }
  { }
)
( 2 0 )
(
  "Jim Donnelly" 3.1416 5 3855 "jimd@cv.hp.com"
)
DUP INFORM
»
```



DEPARTMENT LIST			
NAME:	"Jim Donnelly"		
EMP#:	3.1416	BLDG:	5
PHONE#:	3855		
EMAIL:	"jimd@cv.hp.com"		
ENTER NAME			
EDIT		CANCEL OK	

The CHOOSE Command. A choose box provides an attractive “dialog box” style alternative to menus for selecting from a finite series of options. The parameters supplied to CHOOSE are:

“Title” { Data } start_row → object 1 *or* 0

“Title” The choose box title specified as a string.

{ obj₁ ... obj_n } The objects to be displayed. If each object in the list is a 2 object list, then the first element in each list is displayed in the choose box and the corresponding second object is returned.

start_row The highlight is positioned at the row specified by a real number. If this value is 0, the objects in the list may only be viewed (with no highlight) and 0 will be returned.

If the choose box is terminated by pressing or , the highlighted object (or its corresponding return object) is returned in level 2 and 1 is returned in level 1. If the choose box is terminated by pressing or , the number 0 is returned to the stack.

Example: The program CHOS1 displays a choose box which returns the object highlighted when or is pressed.

CHOS1 123.5 Bytes Checksum #442Ah

```

«
  "CHOOSE A COLOR:"
  { "RED" "ORANGE" "YELLOW" "GREEN"
    "BLUE" "INDIGO" "VIOLET" }
  1 CHOOSE
»

```



Example: The program CHOS2 calculates the weight of an object on another planet. After prompting for the weight of the object on earth, the program displays a choice of planets. The choose box returns a list containing the planet's name and a number corresponding to the force of gravity relative to the earth. The new weight is then calculated using the number and tagged with the planet's name.

CHOS2 429.5 Bytes Checksum #994Bh

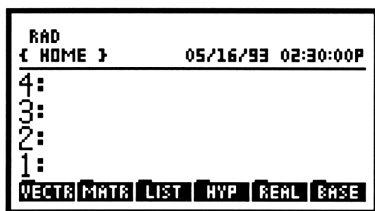
```

«
"Weight on Earth?" "" INPUT OBJ→
CLLCD
"CHOOSE A PLANET:"
(
  ( "Mercury" ( "Mercury" .38 ) )
  ( "Venus"   ( "Venus" .89 ) )
  ( "Mars"    ( "Mars" .38 ) )
  ( "Jupiter" ( "Jupiter" 2.54 ) )
  ( "Saturn"  ( "Saturn" 1.07 ) )
  ( "Uranus"  ( "Uranus" .8 ) )
  ( "Neptune" ( "Neptune" 1.2 ) )
)
1
IF CHOOSE
THEN
  OBJ→ DROP
  ROT *
  "Weight on " ROT + →TAG
END
»

```

Displaying Results

The stack display is organized into three areas:



Status (Area 1)

Stack/Command-line (Area 2)

Menu (Area 3)

The *status area* is contained in the top 16 pixel rows of the display, the *stack/command-line area* is contained in pixel rows 17–56, and the *menu area* occupies the last 8 pixel rows of the display.

The stack display can be modified with the commands CLLCD, DISP, →LCD, and FREEZE.

The CLLCD Command. The command CLLCD clears the stack display. If you execute CLLCD in a program, you may wish to use a FREEZE command to prevent the display from being redrawn until a key is pressed.

The DISP Command. DISP takes an object from level 2, a display line number from level 1, and displays the object on one of 7 logical lines of the display using the medium size font (see the example below and *Converting Text to Graphics*). Line numbers less than 1 are interpreted as 1; line numbers greater than 7 are interpreted as 7. DISP blanks the specified logical line in the display, then displays as much of the object as will fit in the line. If the object extends beyond 22 characters, the first 21 characters are displayed, followed by an ellipsis character (“...”). Objects like programs or matrices and strings with embedded newline characters (character 10) can be shown on more than one line of the display with the DISP command.

The FREEZE Command. FREEZE is used to freeze one or more display areas until a key is pressed. Each display area is represented by a bit in the input value to FREEZE. The values used to identify each of the display areas are:

Display Area Encoding		
Bit	Value	Area
0	1	Status area
1	2	Stack/command-line area
2	4	Menu area

To freeze more than one display area with a single FREEZE command, simply add up the values for each area to freeze. For example, to freeze the status and stack areas, execute 3 FREEZE. To freeze the entire display, execute 7 FREEZE.

Example: The following program illustrates each of the 7 logical display lines associated with the DISP command:

LINES 65.5 Bytes Checksum #85A9h

```

«
  CLLCD                               Clear the display
  1 7 FOR i                           Loop for each line
    "Line " i +                       Build the text for the line
    i DISP                            Display the text
  NEXT                                End of loop
  7 FREEZE                            Freeze the status and stack areas
»

```

Line	1
Line	2
Line	3
Line	4
Line	5
Line	6
Line	7
VECTR MATR LIST WYP REAL BASE	

The →LCD Command. The command →LCD may be used to place a graphics object into the stack display. The graphics object will replace the top 56 lines of the display.

Example: The following program converts an algebraic expression into a graphics object and displays the result in the stack display. *PICT* will remain unchanged.

ALGSTK 54.5 Bytes Checksum #5039h

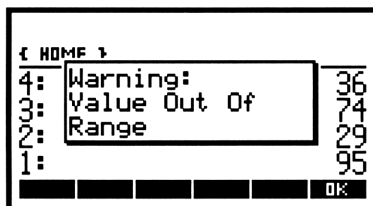
```

«
  CLLCD                      Clear the display
  'SIN(X)/X' 0 →GROB         Make graphic representation of eqn
  →LCD                       Place into stack display
  3 FREEZE                   Freeze the display
»

```

The counterpart to →LCD is LCD→. This command places a copy of the stack display onto level 1 of the stack as a 131x64 pixel graphics object. Note that this is larger than the graphics object that can be displayed with →LCD. To display the entire result of LCD→, store the graphics object in *PICT*, then execute PVIEW, GRAPH, or PICTURE.

The MSGBOX Command. The MSGBOX command displays a string in a box and waits for a , , or keystroke. MSGBOX will divide text between lines at word boundaries, but the string may contain newline characters to format a message that has up to five lines. Messages are displayed in the medium font, and can display up to fifteen characters per line.



If you wish to produce a fancier result, you can add a graphic to a message box – see *LIBEVAL* in *System Programming*.

Recursion

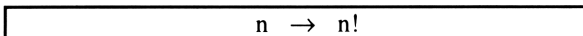
Three conditions must be met to permit recursive programming:

- The system must have an unlimited return stack.
- The system must have an unlimited data stack.
- Programs must be able to call themselves.

The HP 48's data stack and return stack are limited only by available memory, so *recursive programming* is a technique that is available for some forms of problem solving. A recursive program uses a technique for repetitive calculation that works by breaking a problem into smaller pieces and calling itself for each piece.

Factorial Example. The most common illustration of recursive programming is the factorial calculation: $n! = n * (n-1) * (n-2) \dots 2 * 1$, which repeats until $n=1$. The test for completion is to see if the input parameter $n \leq 1$. The program FACTRL (and TREE in *Example Programs*) uses recursion:

FACTRL 85.5 Bytes Checksum #91DEh



```
«  
  → n  
  «  
    IF n 1 ≤ THEN 1  
    ELSE n 1 - FACTRL n *  
    END  
  »  
»
```

List Processing

The HP 48G/GX has list processing capabilities that can simplify some programming tasks.

Command Extensions. Commands that do not accept a list as one of their arguments may take arguments in lists. In the simplest case, { 2 4 5 } SF sets flags 2, 4, and 5. If the command returns a result, the results from a list of arguments will be returned in a list. For instance, { 3 4 } SQ returns { 9 16 }. If a command takes two arguments, the arguments can be supplied in two lists or a list and one object:

$$\begin{aligned} \{ 2\ 3 \} \{ 4\ 5 \} * &\rightarrow \{ 8\ 15 \} \\ \{ A\ B \} 'X' / &\rightarrow \{ 'A/X' 'B/X' \} \\ 'X' \{ A\ B \} / &\rightarrow \{ 'X/A' 'X/B' \} \end{aligned}$$

The command extensions do not apply to commands that have zero or a variable number of arguments (like TEXT or DROPN), nor do they apply to commands that accept any kind of argument (like DUP). Program control structures like START do not accept arguments in lists. The command + is defined to concatenate lists, so the ADD command has been provided for element-wise addition.

Generalized List Processing. The command DOLIST generalizes the command extensions discussed above. A command, program, user-defined function, or named object can be applied to a list of arguments. A command, user-defined function, or named object that requires n arguments can also be applied to n lists of arguments. If n>1 then the lists must all be of the same size. If the number of arguments cannot be inferred from the command or user-defined function the number of arguments must be specified.

DOLIST takes arguments in the following forms:

$$\begin{aligned} \{ \text{list}_1 \} \dots \{ \text{list}_n \} \text{ object} &\rightarrow \{ \text{results} \} \\ \{ \text{list}_1 \} \dots \{ \text{list}_n \} n \text{ object} &\rightarrow \{ \text{results} \} \end{aligned}$$

DOLIST is handy when you want to perform an operation on a list of objects. For example, suppose you want to convert a series of names into strings without quote marks. This can be done by adding a name to an empty string. Instead of creating a FOR loop, use DOLIST:

```
{ A B X1 Y } 1 « "" + » DOLIST →
{ "A" "B" "X1" "Y" }
```

Generating Lists of Results. The command SEQ can be used to generate a list of results from the repeated execution of an object given a variable, start, end, and step values. (These examples assume flag -3 is clear.)

```
« X 2 * » 'X' 1 3 .5 SEQ → { 2 3 4 5 6 }
« 'A' X * » 'X' 1 3 1 SEQ → { A 'A*2' 'A*3' }
```

Processing Sublists. The DOSUBS command steps through a list of arguments supplying groups of objects in the list as arguments to a command or program.

```
{ list } command DOSUBS → { results }
{ list } n program DOSUBS → { results }
```

For each step through the list the specified number of arguments is supplied to the object. This set of arguments can be called a *frame*, and the position of the frame is defined as the position of the first object used in the list. The current position within the list (the frame number) can be determined with the NSUB command and the last position that will be processed with the ENDSUB command. Consider a program that requires two arguments as it is applied to the list { 1 2 3 4 5 }. As DOSUBS executes, NSUB will step from 1 to 4, and ENDSUB will return 4. The arguments used in each frame are underlined in the following table:

NSUB	Arguments Used
1	{ <u>1</u> <u>2</u> 3 4 5 }
2	{ 1 <u>2</u> <u>3</u> 4 5 }
3	{ 1 2 <u>3</u> <u>4</u> 5 }
4	{ 1 2 3 <u>4</u> <u>5</u> }

DOSUBS is handy for tasks like computing first differences or moving averages.

Example: The command Δ LIST can be duplicated with DOSUBS:

```
{ 1 5 11 25 67 } 2 * SWAP - * DOSUBS → { 4 6 14 42 }
```

Example: Suppose you want to create a list containing the moving average of pairs of numbers, with the first and last numbers of the original list added to the respective ends of the result list. The program MAVG does this using the NSUB and ENDSUB commands to monitor the position in the list.

MAVG 135.5 Bytes Checksum #9B72h

```
*  
  2  
  *  
    CASE  
      'NSUB==1' THEN OVER + 2 / END  
      'NSUB==ENDSUB' THEN SWAP OVER + 2 / SWAP END  
      + 2 /  
    END  
  *  
DOSUBS  
*  
  { 1 2 3 4 5 } MAVG → { 1 1.5 2.5 3.5 4.5 5 }
```

Cumulative Argument Processing. The command STREAM takes the first two elements from a list, applies an object to them, then applies the object to the result and the next element, and so on.

{ list } object STREAM → result

The command Π LIST can be duplicated with STREAM:

```
{ 1 5 11 25 67 } * * * STREAM → 92125
```


Meta-Objects

The term *meta-object* refers to a group of objects and their count that resides on the stack. Since stack operations are by nature very efficient, there are times when decomposing a list onto the stack and performing all operations on the stack will be more efficient than rebuilding the list between operations.

The following display shows a meta-object consisting of three names and their count:

{ HOME }	
4:	"STUART"
3:	"KATHRYN"
2:	"FREDERIC"
1:	3
VECTA META LIST NYP REAL BASE	

The term *meta-stack* refers to a group of objects on the stack, some of which may be meta-objects. The term *position* is used instead of *level* when discussing meta-stacks, because a meta-object actually occupies multiple stack levels.

The following meta-stack consists of the string "JANET" in position 1, and meta-objects in positions 2 and 3:

"LB" "JKH" "WMJ" "JS" "EV" 5	21 5 71 3	"JANET"	→
Position 3	Position 2	Position 1	

Notation. To simplify discussions about meta-objects, the following notation is presented. The count is always assumed to be below the elements on the stack. The following symbols are used to indicate objects and meta-objects on the stack, where the right-most element is at the bottom of the stack:

< >	An empty meta-object on the stack (which is just a 0, because the meta-object must have a count).
< ... >	An arbitrary meta-object on the stack.
< Ob ₁ Ob ₂ Ob ₃ >	A meta-object composed of three objects.
< ... > Ob	An object in level 1 and a meta-object that begins in level 2.
< Ob ... >	A meta-object on the stack, with Ob at the head. The head is the element farthest from the count. This is equivalent to the decomposition of the list { Ob ... }.
< ... Ob >	A meta-object on the stack, with Ob at the tail. The tail is the element closest to the count. This equivalent to the decomposition of the list { ... Ob }.
< meta ₂ > < meta ₁ >	Two meta-objects on the meta-stack.

Utility Names. A collection of short utility programs which manipulate meta-objects is presented below. The names start with M, for Meta-object, and use the following naming convention:

- A Refers to the addition of an object to a meta-object.
- D Refers to the deletion of an object from a meta-object.
- M Refers to a meta-object.
- L Refers to a list.
- H Refers to the head of a meta-object.
- T Refers to the tail of a meta-object.
- Z Refers to an empty meta-object.
- 2 Refers to the meta-object in position 2.
- The phrase “to” (converting *to* another form).

Utilities. To establish an empty meta-object on the stack, just place a zero in level 1. To convert a list or vector into a meta-object, execute OBJ→. To convert a meta-object back to a list, execute →LIST. To convert a meta-object back to a vector, execute →ARRAY.

There are many possible routines for meta-object manipulation. The following utility programs are provided to suggest the possibilities. Note that there is no error checking!

MAT	25 Bytes	Checksum #3538h
Adds an object to the tail of a meta-object		
< ... > Ob → < ... Ob >		
« SWAP 1 + »		

MAT2	53.5 Bytes	Checksum #546Eh
Adds an object to the tail of the meta-object in position 2		
< meta ₂ > < meta ₁ > Ob → < meta ₂ Ob > < meta ₁ >		
« OVER 3 + ROLLD DUP 2 + ROLL 1 + OVER 2 + ROLLD »		

MAH	32.5 Bytes	Checksum #4F86h
Adds an object to the head of a meta-object		
< ... > Ob → < Ob ... >		
« OVER 2 + ROLLD 1 + »		

MAH2	66 Bytes	Checksum #1CACH
Adds an object to the head of the meta-object in position 2		
< meta ₂ > < meta ₁ > Ob → < Ob meta ₂ > < meta ₁ >		
« OVER DUP 4 + PICK + 3 + ROLLD DUP 2 + ROLL 1 + OVER 2 + ROLLD »		

MAM2	31 Bytes	Checksum #FAD4h
Concatenates two meta-objects $\langle \text{meta}_2 \rangle \langle \text{meta}_1 \rangle \rightarrow \langle \text{meta}_{1+2} \rangle$		
« DUP 2 + ROLL + »		

MDH	32.5 Bytes	Checksum #813Dh
Extracts an object from the head of a meta-object $\langle \text{Ob} \dots \rangle \rightarrow \langle \dots \rangle \text{Ob}$		
« 1 - DUP 2 + ROLL »		

MDH2	68.5 Bytes	Checksum #BE54h
Extracts an object from the head of the meta-object in position 2 $\langle \text{Ob}_1 \text{Ob}_2 \text{Ob}_3 \rangle \langle \dots \rangle \rightarrow \langle \text{Ob}_2 \text{Ob}_3 \rangle \langle \dots \rangle \text{Ob}_1$		
« DUP 2 + PICK OVER + 2 + ROLL OVER 3 + ROLL 1 - 3 PICK 3 + ROLLD »		

MDT	25 Bytes	Checksum #5F4Dh
Extracts an object from the tail of a meta-object $\langle \dots \text{Ob} \rangle \rightarrow \langle \dots \rangle \text{Ob}$		
« 1 - SWAP »		

MDT2	56 Bytes	Checksum #A95Ch
Extracts an object from the tail of the meta-object in position 2 $\langle \text{Ob}_1 \text{Ob}_2 \text{Ob}_3 \rangle \langle \dots \rangle \rightarrow \langle \text{Ob}_1 \text{Ob}_2 \rangle \langle \dots \rangle \text{Ob}_3$		
« DUP 3 + ROLL OVER 3 + ROLL 1 - 3 PICK 3 + ROLLD »		

ML2M	36 Bytes	Checksum #BF3h
Converts lists in levels 1 and 2 into meta-objects $\{ list_2 \} \{ list_1 \} \rightarrow < meta_2 > < meta_1 >$		
<pre>« SWAP OBJ→ DUP 2 + ROLL OBJ→ »</pre>		

MM2L	36 Bytes	Checksum #499Ah
Converts two meta-objects into lists $< meta_2 > < meta_1 > \rightarrow \{ list_2 \} \{ list_1 \}$		
<pre>« →LIST OVER 2 + ROLLD →LIST SWAP »</pre>		

MSWAP	73.5 Bytes	Checksum #C18Fh
Exchanges two meta-objects $< meta_2 > < meta_1 > \rightarrow < meta_1 > < meta_2 >$		
<pre>« DUP 2 + PICK OVER + 2 + → n « 1 OVER 1 + START n ROLLD NEXT » »</pre>		

MZ2	30 Bytes	Checksum #5038h
Places an empty meta-object in meta-stack position 2 $< meta_1 > \rightarrow < > < meta_1 >$		
<pre>« Ø OVER 2 + ROLLD »</pre>		

Related Commands: DEPTH, DOLIST, DROPN, DUPN, LIST→, →LIST, OBJ→, PICK, ROLL, and ROLLN.

Example: Reversing a List. The following program expects a list as input and returns the reversed list as output. LREV uses a technique similar to that used by the HP 48's REVLIST command.

LREV 57.5 Bytes Checksum #D8C1h

$\{ \text{obj}_1 \dots \text{obj}_n \} \rightarrow \{ \text{obj}_n \dots \text{obj}_1 \}$

«

```
Ø SWAP OBJ→
DUP 1 SWAP
START
  MDT MAT2
NEXT
DROP →LIST
```

»

Example: Filtering a List. The following program expects a list as input and returns a list of all string objects in the list in their original order:

SFILT 81 Bytes Checksum #F4AFh

$\{ \text{obj}_1 \dots \text{obj}_n \} \rightarrow \{ \text{"string}_1 \dots \text{"string}_n \}$

«

```
Ø SWAP OBJ→
DUP 1 SWAP
START
  MDT IF DUP TYPE 2 SAME
  THEN MAH2
  ELSE DROP
END
NEXT
DROP →LIST
```

»

Example: Searching a Vector. The following program scans an input vector and returns two lists: one with numbers $\leq .5$ in level 2, and one with the remaining numbers in level 1:

VSCAN 105.5 Bytes Checksum #3418h

[vector] → [numbers \leq .5] [remaining numbers]

```
«
  Ø SWAP OBJ→ OBJ→ DROP
  DUP 1 SWAP
  START MDT
    IF DUP .5 >
      THEN MAH2
      ELSE MAH
    END
  NEXT
  →LIST
  OVER 2 + ROLLD →LIST
»
```

Example Programs

The example programs in this chapter are intended to be both educational and entertaining – they seek to illustrate various techniques for programming the HP 48. The HP 48 command set is rich enough that there is usually more than one way to do everything.

You can use these programs as a “jumping off” point for your own experiments. Sometimes it’s fun to modify an example to use a different technique – look for modifications that make an example program smaller, faster, or use less memory (or all three!).

Greatest Common Divisor

The program GCD uses a WHILE...REPEAT...END structure to compute the greatest common divisor of two integers a and b . The remainder $r = \text{MOD}(a,b)$ is calculated, b is replaced by a , and a replaced by r until $r=0$.

GCD 42.5 Bytes Checksum #EBD4h

a b → GCD

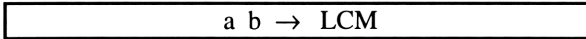
```
⌘
  WHILE DUP                               Loop until r=0
  REPEAT
    SWAP
    OVER MOD                             Calculate r=MOD(a,b)
  END
  DROP                                    Drop r
⌘
```

Example: The G.C.D. of 75488266 and 32565428 is 14.

Least Common Multiple

The program LCM can be used to compute the least common multiple m of two numbers a and b . LCM uses the formula $m=a*b/GCD(a,b)$:

LCM 31.5 Bytes Checksum #EF3Ah



⌘

DUP2 GCD / *

⌘

Example: The L.C.M. of 325 and 340 is 22100.

Square Root's Partial Quotients

The program SQPQ uses an IF...THEN...ELSE...END structure and a DO...UNTIL...END loop to compute a list of partial quotients of the continued fraction equal to the square root of an integer. The list consists of an integer followed by the repeating quotient sequence.

Example: 18 SQPQ returns { 4 4 8 }, which means $SQRT(18) = 4 + 1/(4 + 1/(8 + 1/(4 + 1/(8 + \dots$ with 4 8 repeating.

Example: 95 SQPQ returns { 9 1 2 1 18 } which means $SQRT(95) = 9 + 1/(1 + 1/(2 + 1/(1 + 1/(18 + \dots$ with 1 2 1 18 repeating.

Example: 25 SQPQ returns { 5 }, which means it's exactly 5.

$n \rightarrow \{ \text{list} \}$

```

«
IF DUP √ DUP FP
THEN
  0 1 1 → n sqrt numerator denominator size
  «
    DO
      sqrt numerator + denominator
      / IP DUP denominator *
      'numerator' ST0-
      n numerator SQ -
      'denominator' ST0/
      1 'size' ST0+
    UNTIL
      denominator 1 ==
    END
    numerator DUP + size
  »
ELSE SWAP DROP 1
END
→LIST
»

```

Polynomial Curve Fitting

The program PFIT generates the coefficients for a polynomial which passes through a given set of points. The data is supplied as a matrix, where each row contains the x and y coordinates of a data point.

PFIT 205.5 Bytes Checksum #AF57h

```
«
  DUP SIZE 1 GET           Find number of data points
  → a s
  «
    1 s                     Make array of y-values
    FOR j
      a { j 2 } GET
    NEXT s →ARRY
    1 s                     Make matrix of x-powers
    FOR j
      a { j 1 } GET
      s 1 - 0
      FOR k
        DUP k ^ SWAP
        -1 STEP
      DROP
    NEXT
    { s s } →ARRY
    SWAP DUP2 OVER /       Solve system of equations
    DUP 5 ROLLD RSD       Add RSD for greater accuracy
    SWAP / +
  »
»
```

Example: What polynomial passes through $(-2,-3)$, $(0,7)$, $(1,6)$, and $(2,9)$? Use the MatrixWriter to create the data matrix:

$$\begin{bmatrix} -2 & -3 \\ 0 & 7 \\ 1 & 6 \\ 2 & 9 \end{bmatrix}$$

PFIT returns $[1 -1 -1 7]$, representing the equation $x^3 - x^2 - x + 7$.

Since there is considerable risk involved in forcing a polynomial fit to experimental data, you might be interested to see what the function looks like. The program SHWP takes a matrix containing the data from the stack, plots the data, and then draws the function result from PFIT.

SHWP 277.5 Bytes Checksum #4804h

«

```
STOΣ
ERASE ( #0 #0 ) PVIEW
MINΣ V→ MAXΣ V→
ROT OVER -
1.24 * OVER + YRNG
OVER - 1.03 *
OVER + X RNG
1 NΣ
FOR j
  PICT
  ΣDAT DUP ( j 1 ) GET
  SWAP ( j 2 ) GET R→C
  GROB 5 5 F070709001
  REPL
NEXT
RCLΣ PFIT
OBJ→ 0 SWAP OBJ→ + 2
FOR j
  'X' * j ROLL +
  -1 STEP
STEQ
FUNCTION DRAW
DRAX LABEL
( ) PVIEW
```

»

Save the data in ΣDAT

Show PICT

Find min and max data values

Adjust y range for menu area

Set plot ranges

Loop for number of data points

Get x coordinate

Get y coordinate, make user-unit coordinate

Add arrow grob to PICT

Calculate coefficients

Use Horner's method to create algebraic polynomial

Save equation in EQ

Draw the function

Draw and label axes

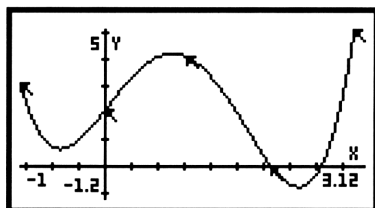
*Show PICT, wait for **CANCEL***

After SHWP has executed, the equation is in *PICT* and the data is stored in ΣDAT .

Example: Find and display the curve that passes through the data points $(-1,3)$, $(0,2)$, $(1,4)$, $(2,0)$, and $(3,5)$. Use the MatrixWriter to create the data matrix:

$$\begin{bmatrix} -1 & 3 \\ 0 & 2 \\ 1 & 4 \\ 2 & 0 \\ 3 & 5 \end{bmatrix}$$

Now run SHWP:



Slope of a Line

The program SLOPE provides a simple example of an application that can be created using the INFORM and MSGBOX commands. A WHILE...REPEAT...END loop runs for as long as INFORM returns 1 (when is pressed). Each time INFORM returns with a new set of values the slope and intercept are calculated. When is pressed INFORM returns 0, and the program ends.

SLOPE 751 Bytes Checksum #7A6h

```
«
0 0 0 0 0 0 → x1 x2 y1 y2 m b
«
WHILE
  "EQUATION OF A LINE"
  (
    ( "X1:" "ENTER X COORDINATE OF POINT 1" 0 )
    ( "Y1:" "ENTER Y COORDINATE OF POINT 1" 0 )
    ( "X2:" "ENTER X COORDINATE OF POINT 2" 0 )
    ( "Y2:" "ENTER Y COORDINATE OF POINT 2" 0 )
    ( "M:" "SLOPE OF LINE" 0 )
    ( "B:" "INTERCEPT" 0 )
  )
  ( 2 3 )
  x1 y1 x2 y2 m b 6 →LIST DUP
  INFORM
REPEAT
  OBJ→ 3 DROPN
  'y2' STO 'x2' STO 'y1' STO 'x1' STO
  CASE
    x1 x2 == y1 y2 == AND
    THEN "Equal Points!" MSGBOX END
    y2 y1 ==
    THEN "Horizontal line Y=" y1 + MSGBOX END
    x2 x1 ==
    THEN "Vertical line X=" x1 + MSGBOX END
    y2 y1 - x2 x1 - / 'm' STO y1 x1 m * - 'b' STO
  END
END
»
»
```

User Interface for User-Defined Functions

Since user-defined functions have a fixed structure, it's fairly simple to isolate the input variables and create a “mini-application” featuring an input form user interface. The syntax of a user-defined function must be exactly:

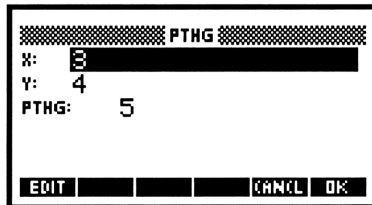
`« → local-names defining-procedure »`

The defining procedure can be either a program or algebraic object, so we know that the variable names reside between the `→` character and the first `«` or `'` delimiter. The function is converted to a string, and the `SUB` command is used to extract the names in string form. To convert the string to a list of names, a `{` is prepended to the string, then the string is converted to a list using `OBJ→`.

Example: Build an application for the user-defined function PTHG:

`« → x y '√(x^2+y^2)' »`

Place the name ‘PTHG’ on the stack and execute UDFUI. Enter values for the input fields x and y , then press `OK`. The result is placed into the PTHG field, and the input form is restarted:



The screenshot shows a graphical user interface for the PTHG function. At the top, a title bar contains the text "PTHG". Below the title bar, there are three input fields. The first field is labeled "x:" and contains the value "3". The second field is labeled "y:" and contains the value "4". The third field is labeled "PTHG:" and contains the value "5". At the bottom of the form, there are four buttons: "EDIT", "CANCEL", "OK", and "OK".

The program continues until you press `CANCEL` or `CANCEL`.

```

«
( ) → name infdat
«
    name RCL
    →STR 4 OVER SIZE SUB          Remove leading « →
    "'«" + 1 OVER "' " POS       Find ' or «
    3 PICK "«" POS MIN 2 - SUB    Extract variables
    "{" SWAP + OBJ→              Convert to list of names
    "" name +                     Create title string
    SWAP name +                   Add fn name to list to
                                   create result field
    1 « ":" + »                  Convert names to field
    DOLIST                        labels with ":"
    DUP SIZE 4 / CEIL OVER        Calculate number of cols
    1 « DROP ( NOVAL ) HEAD »
    DOLIST                         Create default values
    DUP 5 ROLLD                   Save copy of default vals
    4 →LIST                       Save parameters in infdat
    'infdat' STO
    WHILE                          Loop until INFORM
        infdat LIST→ DROP 5 ROLL returns a zero
        INFORM
    REPEAT
        DUP OBJ→ DROP2           Put parameters on stack
        name EVAL                 Evaluate function
        OVER SIZE SWAP PUT       Put result into data list
    END
»
»

```


Sorting Directories

Sorting the variables in a directory is as easy as executing the sequence « VARS SORT ORDER », but the program DSORT adds a useful touch – subdirectories are placed at the beginning of the menu. This program illustrates uses of the commands DECR, GETI, INCR, and PUT. Two handy ways to use DSORT are to assign it to a key or store it in port 0, then you can execute :0:DSORT from any directory.

DSORT 356.5 Bytes Checksum #18C2h

```
«
CLLCD                               Clear display.
"Sorting directory"
1 DISP PATH 2 DISP                 Display current path.
"Sorting VAR list..." 4 DISP     NOTE: "..." is chr_31
VARS SORT                          Sort variables.
"Moving directories..." 5 DISP
DUP DUP SIZE 0 → n m              Level 3 list is target,
«                                  level 2 list is source,
  1 1 n START                     and level 1 is list index.
  GETI                           Get name, increment index.
  IF DUP VTYPE 15 ==             If directory, put into target
    THEN 4 ROLL 'm' INCR         list, increment target count.
    ROT PUT 3 ROLL
  ELSE DROP
  END
NEXT
IF m 0 ≠ THEN                    Level 1 list index wraps to 1
  1 n START                      If directories were moved,
  GETI                          make a 2nd pass to move
  IF DUP VTYPE 15 ≠             the variables into the target
    THEN 4 ROLL 'm' INCR         list.
    ROT PUT 3 ROLL
  ELSE DROP
  END
NEXT
END DROP2                          Drop index and source list.
»
"Executing ORDER..." 6 DISP
ORDER                             Execute ORDER on new list.
»
```

Date Utilities

Some astronomical calculations require dates expressed as a specific number of days after an epoch. Astronomers use the Greenwich mean noon of January 1st 4713 B.C. as this epoch, and the number of days since that epoch (expressed as a fractional quantity) is called the *Julian day number*. Note that since the Julian day begins at 12:00:00 Greenwich time, it is half a day out of step with civil time. The programs JDAYN and JDCAL are based on algorithms in *Practical Astronomy With Your Calculator*, cited in the acknowledgements.

JDAYN. The program JDAYN converts a date to a Julian day number.

JDAYN 361.5 Bytes Checksum #91B7h

year month day → Julian_day_number

```
« → y m d
«
  y m 100 / + d 10000 / +      Date as YYYY.MMDDdd
  IF m 2 ≤
  THEN y 1 - m 12 +
  ELSE y m
  END
  → dat y1 m1
  «
    IF dat 1582.1015 ≥      If date is in Gregorian
    THEN                    calendar, adjust y1, m1
      y1 100 / IP DUP 4
      / IP SWAP - 2 +
    ELSE 0                  Constant B
    END
    365.25 y1 *
    IF y1 0 < THEN .75 - END IP Constant C
    30.6001 m1 1 + * IP      Constant D
    + + d + 1720994.5 +      JD=B+C+D+d+1720994.5
  »
»
»
```

Example: May 16, 1992 is Julian day number 2448758.5.

JDCAL. The program JDCAL converts a Julian day number to a date.

JDCAL 374.5 Bytes Checksum #3B06h

Julian_day_number → year month day

«

.5 + DUP FP SWAP IP	<i>I = integer part, F = fractional part</i>
IF DUP 2299160 >	<i>If in the Gregorian calendar</i>
THEN	
DUP 1867216.25 -	$A = IP(I - 1867216.25) / 36524.25$
36524.25 / IP	
DUP 4 / IP - 1 + +	$B = I + I + A - IP(A/4)$
END	<i>Else B = I</i>
1524 +	$C = B + 1524$
DUP 122.1 - 365.25 /	$D = IP((C - 122.1) / 365.25)$
IP DUP 365.25 * IP	$E = IP(365.25 * D)$
3 PICK OVER - 30.6001	$G = IP((C - E) / 30.6001)$
/ IP 4 ROLL ROT -	<i>Day no.: $C - E + F - IP(30.6001 * G)$</i>
4 ROLL + 30.6001	
3 PICK * IP - SWAP	
IF DUP 13.5 <	<i>Month no.: If $G < 13.5$</i>
THEN 1 -	$m = G - 1$
ELSE 13 -	<i>else $m = G - 13$</i>
END	
ROT	
IF OVER 2.5 >	<i>Year: If $m > 2.5$</i>
THEN 4716 -	$y = D - 4716$
ELSE 4715 -	<i>else $y = D - 4715$</i>
END	
SWAP ROT	

»

Example: Julian day number 2449139.5 is June 1, 1993

The programs DAT2YMD and YMD2DAT convert between dates in HP 48 format (MM.DDYYYY or DD.MMYYYY) and separate numbers for the year, month, and day. Note that negative numbers may be used to represent B.C. dates. For instance, -6.061242 is in 1242 B.C.

DAT2YMD. The program DAT2YMD converts a date in HP 48 format to separate numbers for the year, month, and day.

DAT2YMD 113 Bytes Checksum #AE10h

date → year month day

```

⌘
DUP SIGN SWAP ABS      Determine sign of date, make positive
DUP IP                 Isolate MM (or DD, will decide later)
SWAP FP 100 *          DD.YYYY (or MM.YYYY)
DUP IP                 Isolate DD (or MM)
SWAP FP 10000 *        Isolate YYYY
4 ROLL *               Get sign and adjust year
3 ROLLD                Assume order YYYY MM DD
IF -42 FS?             If -42 set, exchange MM and DD
THEN SWAP
END
⌘

```

Example: 6.011993 → 1993 6 1 (flag -42 clear)

YMD2DAT. The program YMD2DAT assembles separate numbers for the year, month, and day into a date in HP 48 format.

YMD2DAT 95.5 Bytes Checksum #F806h

year month day → date

```

«
IP                               Use only integer part of the day number
IF -42 FS?                       Assert YYYY DD MM order
THEN SWAP
END
100 / +                           MM.DD (or DD.MM)
SWAP
DUP SIGN SWAP ABS                 Get sign of year and use absolute value
1000000 / ROT +                   0.00YYYY + MM.DD to get MM.DDYYYY
*                                 Multiply by sign of year
»

```

Example: 1994 5 18 → 5.18.1994 (flag -42 clear)

DOW. The program DOW finds the day of the week given a date in HP 48 format, based on an algorithm in *Practical Astronomy With Your Calculator*, cited in the acknowledgments.

DOW 167 Bytes Checksum #239Fh

date → "day"

```

«
DAT2YMD                         Convert to year, month, and day
JDAYN                           Calculate Julian day number
1.5 + 7 / FP 7 * -1 RND         Find day number, 0 to 6
{ "Sunday" "Monday"
  "Tuesday" "Wednesday"
  "Thursday" "Friday"
  "Saturday" }
SWAP 1 + GET                     Day of week from 1 to 7
»

```

Example: August 2, 1994 (8.021994) falls on Tuesday.

DAT2STR. The program DAT2STR converts a date in HP 48 format to a string in “MM/DD/YYYY” or “DD.MM.YYYY” format, according to the setting of flag -42. DAT2STR uses the program DAT2YMD.

DAT2STR 272 Bytes Checksum #C8A0h

date → “date”

```

«
RCLF                               Save the system flags, set STD mode
STD                                for number to string conversions
«                                  Subroutine Tostr
  GET                               Get month or day from list
  IF DUP 9 >                       If number is 2 digits
  THEN ""                          then no leading 0,
  ELSE "0"                         otherwise leading 0 required
  END
  SWAP + ←sep +                     Convert to string, add separator
  ROT                              Rotate next list into position
»
→ flg Tostr
«
  DAT2YMD 3 →LIST                  Create { year month day } list
  DUP DUP
  IF -42 FS?                       Check flag -42 for separator chr
  THEN 3 2 ". "
  ELSE 2 3 "/"
  END
  → month day ←sep
  «
    month Tostr EVAL               Create MM/ (or DD.) string
    day Tostr EVAL                 Create DD/ (or MM.) string
    HEAD + +                       Create YYYY string, add MM and DD
  »
  flg STOF                         Restore system flags
»
»

```

Example: 4.161930 DAT2STR → “04/16/1930” (flag -42 clear).

Status Line Animation

TANK. The program TANK recreates a brief moment of Operation Desert Storm. This time, the battlefield appears to be the status area of the display, but is actually a copy of the stack display that is stored in an enlarged *PICT*. Comments appear after the @ character. Note that when you enter the program there is no gap between the characters of grob data in the command line.

[illegible]

«

```

IFERR @ Traps CANCEL key
@ Enlarge PICT so tank can enter from left
# 163d # 64d PDIM
@ Put copy of stack display into PICT
PICT ( # 16d # 0d ) LCD→ REPL
@ Show PICT
( # 16d # 0d ) PVIEW
@ Clear the status area
PICT ( # 16d # 0d ) # 131d # 14d
BLANK REPL
@ Advance tank from left onto battlefield
1 12
START
1 3
FOR i
    PICT x # 4d 2 →LIST tank1 i GET REPL
    # 1d 'x' ST0+
NEXT
NEXT
# 148d 'y' ST0
@ Advance tank from right
1 12
START
1 3
FOR i
    PICT y # 4d 2 →LIST tank2 i GET REPL
    'y' # 1d ST0-
NEXT
NEXT
@ Bang! The thunder of the battlefield ...
200 .02 300 .02 BEEP BEEP
@ Show the shell
PICT ( # 47d # 5d ) shell GXOR
51 115
@ Move shell
FOR i
    PICT i R→B # 5d 2 →LIST shell REPL
4 STEP

```



```

@ Explosion
1 5
FOR i
    PICT { # 114d # 0d } blast i GET REPL
    400 i / .01 BEEP 300 i / .01 BEEP
NEXT
@ Delay
@ (This is a good time to let MEM force garbage collection)
MEM DROP 1 WAIT
@ Move victorious tank off battlefield
1 38
START
1 3
FOR i
    PICT x # 4d 2 →LIST tank1 i GET REPL
    'x' # 1d STO+
NEXT
NEXT
THEN @ End of IFERR trap
END
@ Switch back to the real stack display
TEXT
@ Restore original PICT
pictsav PICT STO
»
»

```

TRAIN. The program TRAIN features a steam train complete with whistle and billowing smoke. Notice how the smoke dissipates! The smoke coordinates are saved on the stack, and cleared at random. Note that when you enter the program there is no gap between the characters of grob data in the command line.

TRAIN 911 Bytes Checksum #A7FDh

«

MEM DROP DEPTH 190 1 FS?C PICT RCL

→ depth × f1 pictsav

«

#191d #64d BLANK

(#60d #0d) LCD→ REPL PICT STO

PICT (#60d #0d) #131d #14d BLANK REPL

(#60d #0d) PVIEW

WHILE ×

REPEAT

PICT × R→B #7d 2 →LIST

GROB 60 7 00E3000000000000E163000000000000CC6100CFFF8FFF30CFF5F7C

AAA8AAA20EFF7F7DFFBAFFF20FFFFFFFFFFFFFFFF70CCC1630303060C00

REPL @ Draw train

IF RAND .23 > THEN @ 77% chance of clearing smoke

IF DEPTH depth - DUP @ If there's smoke on the stack

THEN RAND * ROLL PIXOFF @ then clear one smoke pixel

ELSE DROP END

END

'x' 1 STO- @ Move train left one pixel

IF 1 FS?C THEN @ If flag 1 set, emit additional smoke

× #4d + #7d 2 →LIST PIXON

× #6d + #5d 2 →LIST DUP PIXON @ Save on stack

× #7d + #5d 2 →LIST DUP PIXON @ Save on stack

END

IF RAND .98 > × 60 > AND @ 2% chance to blow whistle

THEN 2112 DUP .28 BEEP .2 WAIT .4 BEEP

END

IF RAND .8 > THEN @ 20% chance of emitting smoke

1 SF @ Request more smoke after train moves

× #4d + #7d 2 →LIST PIXON

× #4d + #6d 2 →LIST DUP PIXON @ Save on stack

× #5d + #6d 2 →LIST DUP PIXON @ Save on stack

END

END

DEPTH depth - DROPN @ Drop remaining smoke

1 IF f1 THEN SF ELSE CF END @ Restore flag 1

TEXT pictsav PICT STO @ Restore original *PICT*

»

»

Customizing the Solver

In *Menus* the options available for menu customization are described. You can create a fairly sophisticated calculating environment using custom menu definitions. While the HP 48G/GX models have HP's best compound interest software built right in, the Time-Value-of-Money (TVM) application remains a classic illustration of what a custom menu definition can do.

The programs TVMCALC, MNU%, and CHOS% mimic some of the most used functions in HP business calculators. Since many variables are used by the equations, you might wish to make a directory BCALC to hold your equations and variables. The programs MNU% and CHOS% illustrate two methods for selecting an application. To reduce directory clutter, these examples are written as single programs.

Compound Interest Calculations. The program TVMCALC implements the classic equation for TVM calculations. The equation supports compound interest calculations for applications where identical payments occur over regular periods which coincide with the compounding periods.

$$0 = PV + PMT \left[\frac{1 - (1 + I)^{-N}}{I} \right] + FV (1 + I)^{-N}$$

The variables are defined as follows:

N The number of periods of the loan or investment.

I The periodic interest rate. If the loan is stated as 8% per year, and payments are monthly, then $I \approx .67$.

PV The present value of the loan or investment. In the case of a loan, this is how much you borrow.

PMT The periodic payment.

FV The future value of the loan or investment. The future value of a loan that is to be paid in full by regular payments is zero.

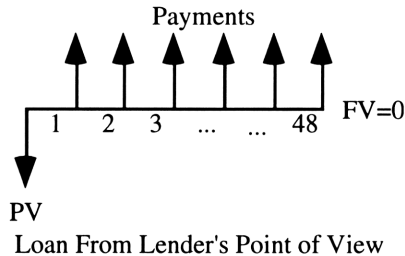
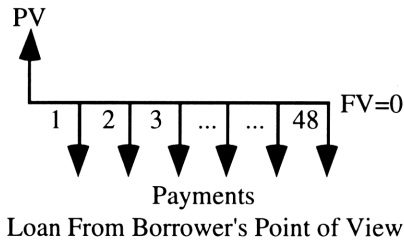
To make the calculations more applicable to the real world, three enhancements have been made to the basic equation:

- Payments can be made at the beginning or end of a period, so the variable *PmtMode* is used to indicate the mode. *PmtMode* is 1 if payments are made at the beginning of the period (called *Begin Mode*), or 0 if payments are made at the end of the period (called *End Mode*).
- In some loans or investments, payments are made on different schedules than one per month. Interest rates are usually quoted on an annual basis, so the variable *I%YR* stores the annual interest rate, and the variable *PYR* is used to adjust for the number of payments per year.
- The equation has been adjusted to let you enter the interest rate as a whole number, so you can enter 8 for 8% instead of .08.

After these changes have been added, the equation becomes a little more complicated:

$$0 = PV + \left(1 + \frac{I\%YR * PmtMode}{100 * PYR}\right) PMT \left[\frac{1 - \left(1 + \frac{I\%YR}{100 * PYR}\right)^{-N}}{\frac{I\%YR}{100 * PYR}} \right] + FV \left(1 + \frac{I\%YR}{100 * PYR}\right)^{-N}$$

In TVM calculations money received is displayed as a positive number; money paid out is displayed as a negative number. The diagrams on the next page are called *cash flow diagrams*, and help to illustrate the flow of monies in a loan or investment.



The program TVMCALC embodies the entire equation and user interface into a single solver equation. When the equation has been activated within the solver, the following menu keys are available:

N	Store or calculate the number of periods of the loan.
I/YR	Store or calculate the annual interest rate.
PV	Store or calculate the present value.
PMT	Store or calculate the payment.
FV	Store or calculate the future value.
MORE	Display the option menu keys.

The option menu keys do the following:

BEG	Set <i>begin</i> mode.
END	Set <i>end</i> mode.
P/YR	Store the number of payments per year.
SHOW	Display the values of all the variables.
RESET	Clear N, I, PV, PMT, FV. Set 12 pmts/year and <i>end</i> mode.
EXIT	Return to the Solver menu.

The user interface is customized to mimic HP's business calculators, and so has extra code for cosmetic purposes. One way this example could be streamlined would be to place the duplicated code that displays the payment mode and number of payments per year into a separate global variable. As you experiment with TVMCALC, you may wish to add or delete other features to see how they work.

TVMCALC 1386 Bytes Checksum #350h

```
«
{
  '0=PV+(1+I%YR*PmtMode/(100*PYR))*PMT*((1-
    (1+I%YR/(100*PYR))^-N)/(I%YR/(100*PYR)))
    +FV*(1+I%YR/(100*PYR))^-N'
  { N I%YR PV PMT FV
    { "MORE"
      « {
        { "BEG" «
          1 'PmtMode' STO
          "Begin Mode" 1 DISP 1 FREEZE
        »
      }
      { "END" «
        0 'PmtMode' STO
        "End Mode" 1 DISP 1 FREEZE
      »
    }
    { "P/YR" «
      'PYR' STO
      PmtMode "Begin" "End" IFTE
      " Mode" + 1 DISP
      PYR " Payments/Year" +
      2 DISP 1 FREEZE
    »
  }
}
```

```

( "SHOW" «
    PmtMode "Begin" "End" IFTE
    " Mode" + 1 DISP
    PYR " Payments/Year" +
    2 DISP
    "N   = " N + 3 DISP
    "I%YR= " I + 4 DISP
    "PV   = " PV + 5 DISP
    "PMT = " PMT + 6 DISP
    "FV   = " FV + 7 DISP
    7 FREEZE
    »
)
( "RESET" «
    0 'PmtMode' STO
    12 'PYR' STO 0 'FV' STO
    0 'PMT' STO 0 'PV' STO
    0 'I%YR' STO 0 'N' STO
    "End Mode" 1 DISP
    "12 Payments/Year" 2 DISP
    1 FREEZE
    »
)
( "EXIT" «
    0 MENU
    PmtMode "Begin" "End" IFTE
    " Mode" + 1 DISP
    PYR " Payments/Year" +
    2 DISP 1 FREEZE
    »
)
)
TMENU

```

```


      PmtMode "Begin" "End" IFTE
      " Mode" + 1 DISP
      PYR " Payments/Year" + 2 DISP 1 FREEZE
    » @ End of program for MORE label
  } @ End of MORE key definition
} @ Menu end
} STEQ
0 'PmtMode' STO 12 'PYR' STO 0 'FV' STO
0 'PMT' STO 0 'PV' STO 0 'I%YR' STO 0 'N' STO
30 MENU @ Display the SOLVR menu
"End Mode" 1 DISP
"12 Payments/Year" 2 DISP
1 FREEZE
»

```

The **MORE** key is implemented by taking advantage of the Solver's variable list feature. The basic strategy is a heavily customized Solver equation list of the form:

{ 'equation' { variable list } }

Each entry in the variable list can be defined as a variable or as a list containing the label and one or more associated programs. For more details on menu key definitions, refer to the example MENUEX in *Menus*.

To solve TVM problems, enter the values you know and solve for the unknown by pressing  followed by the appropriate key.

To try out an example, consider a loan for a car. The loan amount is \$18,280, the annual interest rate is 13%, and the term is for four years. Payments are to be made monthly, so the number of periods (N) is 48. Assume the HP 48 is in FIX 2 display mode, and begin by executing TVMCALC:

Keys:

TVMC
48 N
13 I%YR
18280 PV
0 FV
[G] PMT

Display:

End Mode
12 Payments/Year
N: 48.00
I%YR: 13.00
PV: 18,280.00
FV: 0.00
PMT: -490.41

Now press MORE SHOW to see all the values at once:

```

End Mode
12.00 Payments/Year
N = 48.00
I%YR= 13.00
PV = 18280.00
PMT = -490.41
FV = 0.00
BEG END P/YR SHOW RESET EXIT

```

Percentage Calculations. The program MNU% consists of a custom menu definition containing four commonly used percentage equations, along with text to remind you which is which. The equations are:

$$\% \text{ change} = \left(\frac{NEW - OLD}{OLD} \right) * 100$$

$$\% \text{ of total} = \left(\frac{PART}{TOTAL} \right) * 100$$

$$\text{Markup as \% of cost} = \left(\frac{PRICE - COST}{COST} \right) * 100$$

$$\text{Markup as \% of price} = \left(\frac{PRICE - COST}{PRICE} \right) * 100$$

When you execute MNU%, the menu and prompts are displayed:

```

Select a function:
%CHG Percent change
%TOT Percent of total
MU%C Markup as % cost
MU%P Markup as % price
%CHG %TOT MU%C MU%P

```

Pressing a menu executes the corresponding program in the menu definition, which loads the appropriate equation into EQ and starts up the solver menu. For instance, press %CHG to display the first equation:

```

EQ: ( '%CHG=(NEW-OLD ...
4:
3:
2:
1:
OLD NEW %CHG EXP1


```

MNU% 683.5 Bytes Checksum #7905h

```
«
{
  { "%CHG" «
    { '%CHG=(NEW-OLD)/OLD*100'
      { OLD NEW %CHG }
    } STEQ 30 MENU
  }
  { "%TOTL" «
    { '%TOT=PART/TOTAL*100'
      { TOTAL PART %TOT }
    } STEQ 30 MENU
  }
  { "MU%C" «
    { 'M%C=(PRICE-COST)/COST*100'
      { COST PRICE M%C }
    } STEQ 30 MENU
  }
  { "MU%P" «
    { 'M%P=(PRICE-COST)/PRICE*100'
      { COST PRICE M%P }
    } STEQ 30 MENU
  }
} TMENU
CLLCD
"Select a function:" 1 DISP
"%CHG Percent change" 3 DISP
"%TOT Percent of total" 4 DISP
"MU%C Markup as % cost" 5 DISP
"MU%P Markup as % price" 6 DISP
3 FREEZE
»
```

Displays menu
Clear display
Show options

Freeze display

Once the equation has been initialized in the Solver, enter the variables you know, then press  and the unknown variable.

The program CHOS% uses a choose box instead of a custom menu to select an equation. The list supplied to the CHOOSE command consists of four lists, each containing a description to be shown in the choose box and the corresponding solver equation to be stored in *EQ*.

CHOS% 521 Bytes Checksum #4015h

```

«
  CLLCD
  "SELECT A FUNCTION: "           Choose box title
  (
    ( "Percent change"
      ( '%CHG=(NEW-OLD)/OLD*100'
        ( OLD NEW %CHG )
      )
    )
    ( "Percent of tot"
      ( '%TOT=PART/TOTAL*100'
        ( TOTAL PART %TOT )
      )
    )
    ( "Mkp as % cost"
      ( 'M%C=(PRICE-COST)/COST*100'
        ( COST PRICE M%C )
      )
    )
    ( "Mkp as % price"
      ( 'M%P=(PRICE-COST)/PRICE*100'
        ( COST PRICE M%P )
      )
    )
  )
  1                               Choose highlight on 1st item
  IF CHOOSE                      If OK pressed...
  THEN
    STEQ                         Store equation
    30 MENU                     Display the solver menu
  END
»

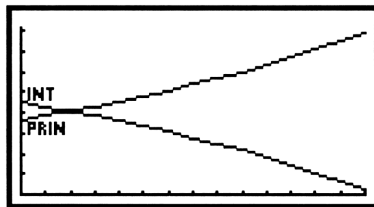
```

Amortization Plot

Given a Time-Value-of-Money problem that's been established in the TVM variables, the command AMORT may be used to calculate the amounts applied to principal and interest and the remaining balance of a loan after a given number of payments. With this function in hand, it's interesting to plot the amounts applied to principal and interest over the life of a loan.

The program AMPLT plots the amounts applied to principal and interest for each year of a loan using the following algorithm:

- Calculate first year's amortization data using PYR periods.
- Store new balance back into PV.
- Save amounts applied to interest and principal during the first year in variables *int* and *prin*.
- For each year:
 - Calculate next year's amortization data, update PV with new balance.
 - Duplicate the new values for *int* and *prin*.
 - Draw lines from last year's *int* to this year's *int* and last year's *prin* to this year's *prin*.
 - Update *int* and *prin*.



```

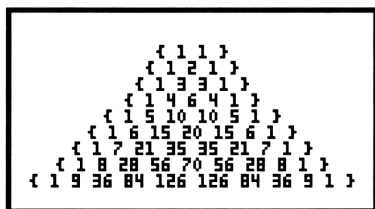
«
{ PICT PPAR } PURGE           Purge PPAR and PICT
CLLCD "Scaling ..." 1 DISP Display status message
PV N PYR /                     Calculate number of years
PYR AMORT                     Amortize first year
'PV' STO                      Store new balance in PV
NEG SWAP NEG                   Principal and interest need to
→ pv years int prin           be positive for plotting
«
  1 years XRNG                 Scale x-axis
  0 PMT NEG PYR * YRNG         Scale y-axis
  (1,0) AXES                   Set axes intersection
  1 10 PMT XPON 1 + ^          Set x, calc y-tick intervals
  2 →LIST ATICK               Set tick intervals
  { #0 #0 } PVIEW             Show new PICT
  DRAX                        Draw axes with tick marks
  PICT
  1 int R→C C→PX OBJ→ DROP Calculate coordinates for INT
  SWAP 3 + SWAP 5 - 2 →LIST
  "INT" 1 →GROB GOR           Put INT label into PICT
  PICT
  1 prin R→C C→PX OBJ→ DROP Calculate coords for PRIN
  SWAP 3 + SWAP 1 + 2 →LIST
  "PRIN" 1 →GROB GOR          Put PRIN label into PICT
  2 years FOR yr              Loop for each year
    PYR AMORT                 Amortize next year
    'PV' STO                  Save new PV
    NEG SWAP NEG DUP2         Positive values for plotting
    yr SWAP R→C               New principal value coord.
    yr 1 - prin R→C           Old principal value coord
    LINE                      Draw line
    yr SWAP R→C               New interest coord
    yr 1 - int R→C            Old interest coord
    LINE                      Draw line
    'prin' STO 'int' STO      Update old prin & int
  NEXT pv 'PV' STO            Loop end, restore original PV
  PICTURE
»
»

```

Pascal's Triangle

Pascal's Triangle is the arrangement of rows of numbers where the n th row consists of the binomial coefficients $\binom{n}{r}$ with $r = 0, 1, \dots, n$. When the numbers are displayed in rows centered about a vertical axis, you can see that $\binom{n+1}{r} = \binom{n}{r-1} + \binom{n}{r}$, which are positioned to the left and right above it. For instance, $\binom{6}{3}$ is 15, which is the sum of $\binom{5}{2}$, which is 5 and $\binom{5}{3}$ which is 10.

The HP 48 display can comfortably display the first nine rows of this arrangement. The program PASCAL on the next page generates the following display:



```

«
  «
    { 1 }
    → prevrow newrow
    «
      1 prevrow SIZE 1 -
      FOR r
        'newrow'
        prevrow r
        DUP 1 + SUB
        OBJ→ DROP +
        STO+
      NEXT
      newrow 1 +
    »
  »
  PICT RCL
  → DoNewRow pict
  «
    PICT PURGE
    { #0d #0d } PVIEW
    { 1 1 }
    1 9
    FOR n
      PICT
      OVER 1 →GROB
      DUP SIZE DROP
      2 / 66 SWAP -
      n #6d *
      2 →LIST SWAP
      REPL
      DoNewRow EVAL
    NEXT DROP
    { } PVIEW
    pict PICT STO
  »
»

```

Subroutine DoNewRow
Initial value of new row

Loop for r subintervals

Extract elements
Add to form new element
Add to end of list in newrow

Add trailing 1

Save old PICT

Purge PICT
Create and show new PICT
First row is { 1 1 }

Loop for 10 rows

Create a copy of row as grob
Find width of grob
Center the grob
Calculate row coordinate
Combine coords into list
Place grob in PICT
Calculate new row value
Complete loop, discard last row
*Show PICT, wait for **CANCEL***
Restore old PICT

Plotting Inequalities

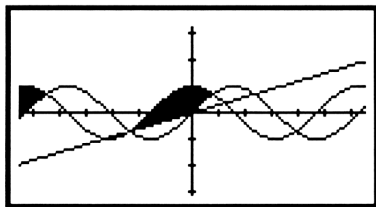
The program INPLOT plots the intersection of a series of two or more inequalities stored in a list in the reserved variable *EQ*. The equations must take the form $y=f(x)$. The program assumes you have set the plot scale in *PPAR*. The program tests the midpoint of vertical lines drawn between the plot boundary and the functions to see if that point satisfies all inequalities. A status pixel blinks along the top of the plot to indicate the progress of the plot.

The program clears flag -3 to assert symbolic results mode. Since at least two equations are being plotted, you might wish to set flag -22 to draw the equations simultaneously.

Example: Plot the logical AND of the equations $Y > \sin(X)$, $Y < \cos(X)$, and $Y > .3 * X$. Store the following list in *EQ*:

```
{ 'Y>SIN(X)' 'Y<COS(X)' 'Y>.3*X' }
```

Assuming that Radians mode is set and *PPAR* contains the default scaling parameters, the program INPLOT may be used to produce the following plot:



«

```
-3 CF RCEQ DUP SIZE
IF DUP 2 <
THEN DROP2 513 DOERR
END
```

*Symbolic results**Require at least 2 equations*

```
{ } { }
```

Values for neweqns and pts

```
PPAR 1 GET C→R
```

Xmin & Ymin

```
PPAR 2 GET C→R
```

Xmax & Ymax

```
OVER 5 PICK -
```

Calculate plot step size

```
PICT SIZE DROP B→R /
```

```
→
```

```
oldeqns numeqns neweqns
```

```
pts xmin ymin xmax ymax
```

```
step
```

«

```
oldeqns 1
```

Convert inequality into

```
« OBJ→ DROP2 = »
```

form $y=f(x)$

```
DOLIST
```

```
DUP 'neweqns' STO STEQ
```

Store in neweqns and EQ

```
ERASE { #0 #0 } PVIEW
```

Show PICT

```
DRAX DRAW
```

Plot equation lines

```
oldeqns 1
```

Extract right side of eqns

```
« OBJ→ DROP2 SWAP DROP »
```

```
DOLIST 'neweqns' STO
```

Store into neweqns

```
xmin xmax
```

Loop from left to right

```
FOR x
```

```
  x ymax R→C DUP
```

Coordinate of top pixel

```
  PIX? DUP2
```

Save coord & pixel state

```
  « PIXOFF » « PIXON »
```

```
  IFTE
```

Toggle pixel

```
  x 'X' STO
```

Update value of X

```
  ymin ymax 2 →LIST
```

Plot borders begin pts list

```
  neweqns 1 « →NUM »
```

Calculate function values

```
  DOLIST + SORT
```

Sort list into ascending order

```
  'pts' STO
```

Save into pts

```
  1 numeqns 1 +
```

Loop for each line segment

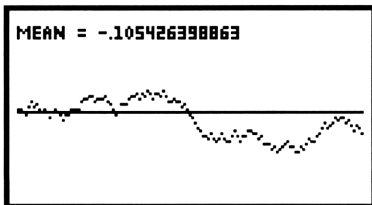
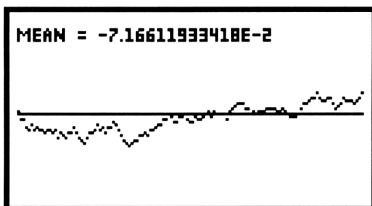
FOR n	
pts n n 1 + SUB	<i>Extract segment endpoints</i>
OBJ→ DROP	
DUP2 + 2 / 'Y' STO	<i>Calculate midpoint</i>
IF	
1 1 numeqns	<i>Loop for each equation</i>
FOR m	
oldeqns m GET	<i>Test inequality, accumulate</i>
→NUM AND	<i>into number on stack</i>
NEXT	
THEN	
× SWAP R→C ×	<i>Draw segment if true for</i>
ROT R→C LINE	<i>all equations</i>
ELSE DROP2	<i>Otherwise discard points</i>
END	
NEXT	<i>End of segment loop</i>
« PIXON » « PIXOFF »	
IFTE	<i>Restore status pixel</i>
step STEP	<i>End of plot loop</i>
PICTURE	<i>Display PICT</i>
»	
»	

Brownian Motion

The Scottish biologist Robert Brown (1773–1858) first observed that tiny particles in a liquid suspension make small, erratic movements. This motion (later dubbed *Brownian Motion*) results from motion and collisions at a molecular level that are proportional to temperature. Two examples are presented here that illustrate Brownian motion.

A Brownian Line. Brownian motion in one dimension can be illustrated with a *Brownian line*. As a line is drawn from left to right, a random incremental vertical step is taken. Each point P_n along the line is defined as $P_{n-1} + S * (RAND - .5)$, where S is a suitable scale factor and $RAND$ is a random number between 0 and 1.

The line could represent the landscape elevations on a straight line between two cities. The line could also represent the cumulative fortunes of two people gambling with each other – “up” motions represent a win for one player, “down” motions represent a win for the other player. Sometimes one player will do very well, sometimes very poorly.



Over the long run, the line should hover around the axis. The program BROWNLN draws the line, uses the statistics matrix ΣDAT to accumulate the values, calculates and displays the mean.

BROWNLN 285 Bytes Checksum #AB17h

«	
PICT PURGE	<i>Purge existing PICT</i>
{ #0 #0 } PVIEW	<i>Create and show default PICT</i>
(0,-1) PMIN (130,1) PMAX	<i>Establish scale (131 pixel width)</i>
(0,0) (130,0) LINE	<i>Draw axis</i>
CLΣ	<i>Purge existing ΣDAT</i>
0	<i>Starting value</i>
0 130	<i>Loop from left to right</i>
FOR x	
RAND .5 - .15 * +	<i>Calculate new value. S=.15.</i>
DUP Σ+	<i>Accumulate new value in ΣDAT</i>
x OVER R→C PIXON	<i>Turn on pixel</i>
NEXT DROP	<i>End loop, drop final value</i>
PICT { #0 #0 }	<i>Coordinates for mean</i>
"MEAN = "	
MEAN + 1 →GROB	<i>Calculate mean, convert to grob</i>
REPL	<i>Place mean into PICT</i>
{ } PVIEW	<i>Show PICT, wait for CANCEL</i>
»	

A Random Walk. Brownian motion in two dimensions can be illustrated with the classic example of an intoxicated person starting in the middle of a large parking lot. From a starting position, each step is taken in a random direction. The distance from the starting position is proportional to the square root of the number steps taken multiplied by the step size.

To calculate the position of step n in Cartesian coordinates, generate a random number d between 0 and 360. Then:

$$X_n = X_{n-1} + \cos(d) \quad Y_n = Y_{n-1} + \sin(d)$$

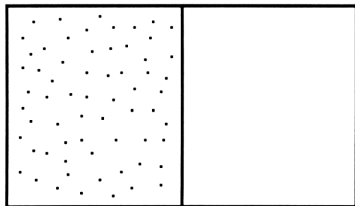
RANWALK 183 Bytes Checksum #1D52h

«	
PICT PURGE	<i>Purge existing PICT</i>
{ #0 #0 } PVIEW	<i>Create and show default PICT</i>
DEG	<i>Assert degrees mode</i>
(-32.5,-15.5) PMIN	<i>Scale PICT</i>
(32.5,16) PMAX	
0 0	<i>Initial values for X and Y</i>
0 250	
START	<i>Loop for 250 steps</i>
RAND 360 *	<i>Random direction d</i>
SWAP OVER SIN +	<i>Add sin(d) to Y</i>
SWAP COS ROT +	<i>Add cos(d) to Y</i>
SWAP DUP2 R→C PIXON	<i>Turn on pixel</i>
NEXT DROP2	<i>End loop, drop X and Y</i>
{ } PVIEW	<i>Show PICT, wait for CANCEL</i>
»	

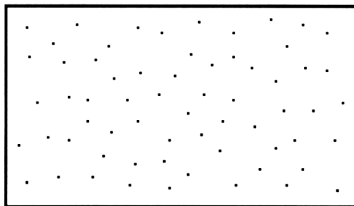


Entropy

Entropy is a measure of disorder in a system, and must increase over time as stated by the second law of thermodynamics. An elegant illustration of this is the classic picture of a box divided into two halves by a removable partition. The free expansion of a gas from the left half across both halves when the partition is removed is a good example of an irreversible process in which the entropy has increased.



Initial Conditions



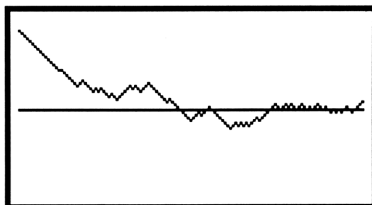
Final Conditions

The progress of the system can be simulated in the HP 48 graphics display. Two programs are presented – one shows the proportion of molecules on each side as a function of time, the other shows the box itself.

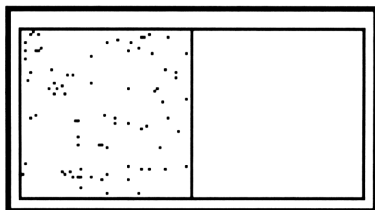
Proportion Over Time. The program ENTROPY plots the proportion of gases between two sides of a box over time after the partition has been removed. The horizontal axis will represent time starting at the left and the vertical axis will show the proportion of molecules on each side of the box. The initial conditions state that there are 64 molecules one side of the box. For each time that a molecule crosses the midline of the box the probability $P_{R \rightarrow L}$ that a molecule will cross from the right side to the left side is given by $N_L / (N_L + N_R) * \text{RAND}$, where N_L is the number of molecules on the left side, N_R is the number of molecules on the right side, and RAND returns a random number between 0 and 1.

For each crossing, the ratio is compared to a new random number between 0 and 1, and if the random number is greater than the ratio, a molecule is said to move from right to left.

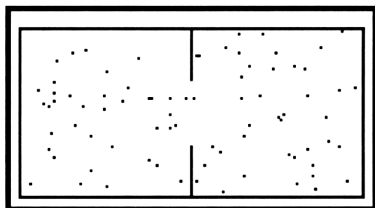
«	
PICT PURGE	<i>Purge existing PICT</i>
{ #0d #0d } PVIEW	<i>Create and show default PICT</i>
(0,1) PMIN	<i>Scale PICT</i>
(130,64) PMAX	
(0,32) (130,32) LINE	<i>Draw midline</i>
64	<i>Initial number of molecules on left</i>
0 130	<i>Loop for 130 crossings</i>
FOR x	
RAND OVER 64 /	<i>Calculate left/right ratio</i>
IF <	<i>If ratio is less than random number</i>
THEN 1 -	<i>Then one molecule moves right</i>
ELSE 1 +	<i>Otherwise one molecule moves left</i>
END	
x OVER R→C PIXON	<i>Show proportion</i>
NEXT DROP	<i>Loop end, drop final value</i>
{ } PVIEW	<i>Display PICT, wait for CANCEL</i>
»	



Gas Simulation. The program FREXP sets up the initial conditions for the experiment. The program begins by drawing 80 molecules, beeping, and waiting for a key:



Now begin the experiment by pressing a key. The molecules will begin to move from one side to the other. The program runs until you press a key.



The WHILE loop in the program continues until a key is pressed. During this time, the state of the system is represented by lists of molecule coordinates for the left and right sides of the box in stack levels 3 and 2, and the count of molecules on the left side in level 1.

```

«
80 PICT RCL → n pictsav          Save PICT
«
#131d #64d BLANK PICT STO        Blank PICT
{ #0d #0d } PVIEW                Show PICT
{ #0d #0d } { #130d #63d } BOX   Draw box
{ #65d #0d } { #65d #63d } LINE Partition
1 n START                        Setup molecules
  RAND 63 * 1 + R→B              X coordinate
  RAND 60 * 1 + R→B              Y coordinate
  2 →LIST DUP PIXON              Turn on pixel
NEXT n →LIST                     Left list created
440 .1 BEEP 0 WAIT DROP          Wait for key
{ #65d #20d } { #65d #43d } TLINE Open partition
{ } n                             Right list, left count
WHILE KEY NOT                    Run until key hit
REPEAT
  IF DUP n / RAND >              If left→right
  THEN
    1 - 3 PICK 1 GET PIXOFF       Turn off pixel
    RAND 63 * 66 + R→B            Create new location
    RAND 60 * 1 + R→B
    2 →LIST DUP PIXON             Turn on new pixel
    ROT SWAP 1 →LIST + SWAP       Store in right list
    ROT 2 OVER SIZE SUB 3 ROLLD   Subtract from left
  ELSE                             Otherwise right→left
    1 + SWAP DUP 1 GET PIXOFF     Turn off pixel
    2 OVER SIZE SUB SWAP ROT       Subtract from right
    RAND 63 * 1 + R→B             Create new location
    RAND 60 * 1 + R→B
    2 →LIST DUP PIXON             Turn on new pixel
    1 →LIST + 3 ROLLD            Add to left list
  END
END
4 DROPN TEXT pictsav PICT STO    Clean up
»
»

```

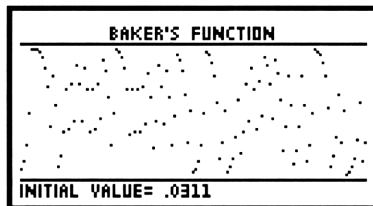
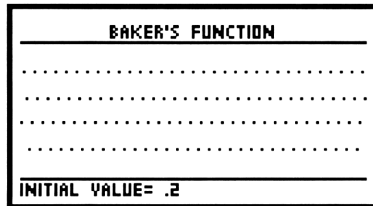
Chaos

In the Brownian line example, we observed a system which is fairly predictable – the results of successive iterations over time live in a known range. There are functions whose domain and range live in the same finite interval whose behavior is sensitive to initial conditions. Sensitivity to initial conditions can be defined by stating that within the domain of a function f the difference between $f(x)$ and $f(x+\Delta)$ after n iterations exceeds some criterion ϵ .

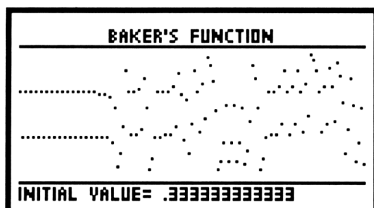
Sensitivity to initial conditions can be illustrated with the baker's function B:

$$B(x) = \begin{cases} 2x & \text{for } 0 \leq x \leq .5 \\ 2x - 1 & \text{for } .5 < x \leq 1 \end{cases}$$

Plotting the Baker's Function. The program BAKER plots successive iterates of B from left to right on a vertical scale from 0 to 1. The following plots illustrate that the behavior of the baker's function can be periodic or apparently random, depending on the initial value.



There is a hazard to putting too much faith in the plots for a large number of iterates. For instance, an initial value of $1/3$ yields the sequence $2/3, 1/3, 2/3, 1/3$, etc. Rounding errors can yield problems as illustrated on the next page.



BAKER 454.5 Bytes Checksum #304Fh

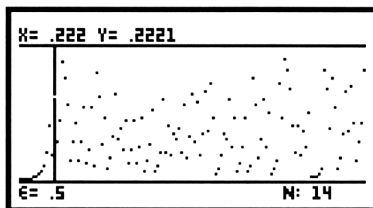
```

«
  PICT PURGE                                Purge existing PICT
  { #0d #0d } PVIEW                         Create and show default PICT
  (1,-.15) PMIN                             Scale PICT
  (131,1.15) PMAX
  PICT { #34d #0d }
  "BAKER'S FUNCTION"
  1 →GROB                                    Create title grob
  REPL                                       Display title
  { #0d #6d } { #130d #6d }                Coordinates for top line
  LINE                                      Draw top line
  { #0d #57d }
  { #130d #57d } LINE                      Draw bottom line
  PICT { #0d #59d }                       Coordinates for value
  "INITIAL VALUE= "
  4 PICK +
  1 →GROB                                    Create value grob
  REPL                                       Display value grob
  1 131 FOR n                               Loop for 131 iterations
    IF DUP .5 ≤                             Baker's function
    THEN 2 *
    ELSE 2 * 1 -
    END
    n OVER R→C PIXON                       Display pixel
  NEXT DROP                                End loop, drop final value
  { } PVIEW                                Show PICT, wait for CANCEL
»

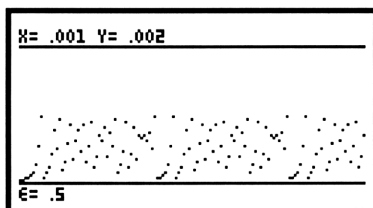
```

Comparing Iterates of Baker's Function. To illustrate the sensitivity of initial conditions, use the program BAKDIFF. Starting with two initial values x and y , $B(x)$ and $B(y)$ are computed at each iterate n , and the difference between the iterates is calculated. If the difference exceeds the value ϵ , a line is drawn and n is displayed. The value for ϵ is defined as .5 and stored in local variable ϵ .

Use BAKDIFF to compare the iterates of $x=.222$ and $y=.2221$:



Sometimes the differences do not exceed your critereon for ϵ , but you still can't reliably predict a value for $B_n(y)$ based on $B_n(x)$:



```

«
  PICT PURGE                                Purge existing PICT
  ( #0d #0d ) PVIEW                        Create and show default PICT
  ( 1, -.15 ) PMIN                         Scale PICT
  ( 131, 1.15 ) PMAX
«
  IF DUP .5 ≤                               Baker's function
  THEN 2 *
  ELSE 2 * 1 -
  END
»
.5                                           Value for ε
→ x y baker ε
«
  ( #0d #6d )
  ( #130d #6d ) LINE                       Draw top line
  PICT ( #0d #0d )                         Coordinates for initial values
  "X= " x +                                Build initial value string
  " Y= " + y +
  1 →GROB                                  Convert to grob
  REPL                                     Display value grob
  ( #0d #57d )
  ( #130d #57d ) LINE                       Draw bottom line
  PICT ( #0d #59d )                         Coordinates for ε
  "ε= " ε +                                Build ε string
  1 →GROB                                  Convert to grob
  REPL                                     Display ε
  1 CF                                     Use flag 1 for line control
  1 131
  FOR n                                     Loop for 131 iterations
    x baker EVAL                           Calculate B(x)
    DUP 'x' STO                           Store new value for x
    y baker EVAL                           Calculate B(y)
    DUP 'y' STO                           Store new value for y
    - ABS                                  Calculate difference
    n OVER R→C PIXON                      Display difference

```

IF 1 FC?	<i>If no line has been drawn</i>
THEN	<i>Then check against ϵ</i>
IF $\epsilon >$	
THEN	
n 0 R→C n 1 R→C TLINE	<i>Draw line if ϵ exceeded</i>
1 SF	<i>Set control flag</i>
PICT { #100d #59d }	<i>Coordinates for n</i>
"N: " n +	<i>Build string for n</i>
1 →GROB REPL	<i>Convert to grob and display n</i>
END	
ELSE DROP	<i>Discard difference if line drawn</i>
END	
NEXT	<i>End of iteration loop</i>
{ } PVIEW	<i>Display PICT, wait for CANCEL</i>
»	
1 CF	<i>Clear control flag</i>
»	

Chaotic Orbits. A model of a particle in a chaotic orbit may be easily programmed on the HP 48. The HP 48 program ORBIT was inspired by the program MIRA in the book *Fractals* by Hans Lauwerier (cited in the acknowledgments). The successive iterates are calculated by:

$$x_{n+1} = y_n - F(x_n)$$

$$y_{n+1} = -bx_n + F(x_{n+1})$$

where:

$$F(x) = ax + \frac{2(1-a)x^2}{1+x^2}$$

The value for a controls the chaotic behavior (orbits are stable when a is 1). The value of b controls the spiral nature of the orbit. If b is just slightly less than 1, the orbit spirals inward.

Ordinarily one might expect that the display of the HP 48 is too small to yield interesting pictures from chaotic functions, but careful selection of initial conditions and scaling parameters can yield many interesting pictures.

ORBIT takes the number of iterates n , initial values for a and b , the starting position x and y , and scaling coordinates from an input form.

ORBIT 528.5 Bytes Checksum #8A6Eh

```

«
"ORBIT PARAMETERS"
{ "ITERATES:" { } "A:" "B:" "X:" "Y:"
  "PMIN:" "PMAX:" } { 2 0 }
{ 0 0 0 0 0 (0,0) (0,0) } DUP
IF INFORM
THEN
  OBJ→ DROP
  PICT PURGE
  { #0d #0d } PVIEW
  PMAX PMIN
  2 5 PICK 2 * -
  3 PICK SQ DUP
  3 PICK *
  7 PICK 6 PICK * +
  SWAP 1 + / 0
  → a b x y c w z
«
  0 SWAP
  FOR n x
    IF n 10 >
    THEN DUP y R→C PIXON
    END
    'z' STO
    b y * w +
    DUP 'x' STO
    a OVER * SWAP SQ
    DUP c * SWAP 1 + / +
    DUP 'w' STO
    z - 'y' STO
  NEXT
»
{ } PVIEW
END
»

```

Purge existing PICT

Create and show default PICT

Scale PICT

Calculate intermediate value

Calculate initial value for w

Create local variables

Loop for n iterations

Plot only after 1st 10 points

Set pixel at (x,y)

Save old x in z

Calculate new x

Calculate new w

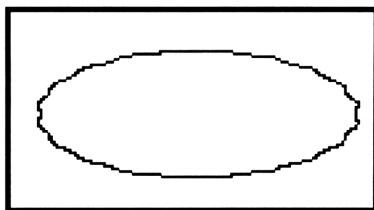
Complete new value for y

Save y

*Show PICT, wait for **CANCEL***

To get acquainted with ORBIT, begin with a somewhat stable orbit:

n	a	b	x	y	PMIN	PMAX
700	.95	1	0	7.5	(-25,-10)	(27,10)



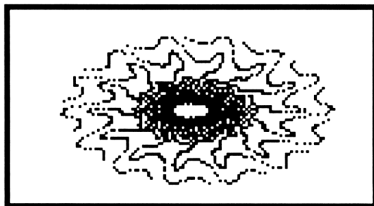
The value of a controls the chaotic behaviour of the orbit. Reduce a to see its effect on the orbit and adjust the scale to keep the picture large:

n	a	b	x	y	PMIN	PMAX
700	.9	1	0	7.5	(-20,-8)	(22,8)



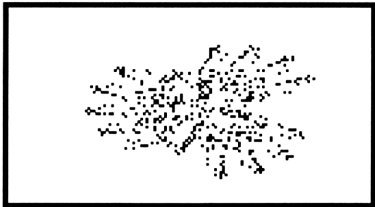
Now reduce b very slightly to make the orbit spiral inward:

n	a	b	x	y	PMIN	PMAX
2200	.9	.998	0	7.5	(-20,-8)	(22,8)



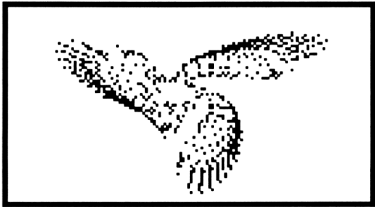
Now try something completely different:

n	a	b	x	y	PMIN	PMAX
600	-.4	.99	4	0	(-12,-10)	(13,10)



or:

n	a	b	x	y	PMIN	PMAX
900	-.48	.935	4.1	0	(-11,-10)	(14,7)



Here's some more to try. Remember that very small variations in initial conditions can result in dramatic changes to the orbit. For instance, try the second example in the table below with values for *a* of $-.24$, $-.25$, and $-.26$.

Chaotic Orbits						
n	a	b	x	y	PMIN	PMAX
500	-.05	.985	9.8	0	(-13,-11)	(17,11)
1000	-.24	.998	3	0	(-12,-10)	(14,10)
1000	.2	1	11	0	(-20,-16)	(22,17)
400	.3	1	8	0	(-35,-19)	(35,19)
500	.4	1	0	5	(-13,-8)	(16,8)

Fractal Trees

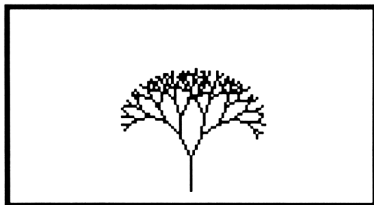
The program TREE draws a fractal tree in *PICT*. This program was adapted from a program written by Carlos Talbot. The use of global variables for subroutines was changed to use local variables, and an INFORM command is used to obtain the plot parameters. The program continues until you press **CANCEL** in the input form. The tree parameters are maintained after the tree is drawn so you can remember the parameters used or change one to see how the tree is affected.

There are six parameters to TREE.

- Left α A number from 1 to 5 that controls the length of left branches.
- Right α A number from 1 to 5 that controls the length of right branches.
- Left \angle Controls the angle of left branches – larger angles produce greater curvature.
- Right \angle Controls the angle of right branches.
- Height The height of the tree trunk in pixels.
- Levels The number of recursion levels. It's best to start out with a small number to see how the tree begins to appear, then add additional levels "to taste".

The tree below was drawn with the following parameters:

Left α	Right α	Left \angle	Right \angle	Height	Levels
2.0	2.2	24	26	12	6



Here's some more sample trees to "grow" in your HP 48 display:

Fractal Tree Samples					
Left α	Right α	Left \angle	Right \angle	Height	Levels
2.0	.00001	55	0	35	10
1.2	1.2	90	90	30	7
3.0	3.0	33	33	13	9
2.0	2.2	20	28	12	10

The program TREE has three small "subroutines" stored in the local variables $\leftarrow Dline$, $\leftarrow Point$, and $\leftarrow Generate$. These subroutines share data with the main program via the stack and compiled local variables.

TREE 2112 Bytes Checksum #6A4h

```

«
CLLCD
( 0 0 0 0 0 0 )
WHILE
  "FRACTAL TREE"
  (
    ( "LEFT  $\alpha$ :" "ENTER LEFT  $\alpha$ " 0 )
    ( "RIGHT  $\alpha$ :" "ENTER RIGHT  $\alpha$ " 0 )
    ( "LEFT  $\angle$ :" "ENTER LEFT  $\angle$ " 0 )
    ( "RIGHT  $\angle$ :" "ENTER RIGHT  $\angle$ " 0 )
    ( "HEIGHT:" "ENTER TRUNK HEIGHT" 0 )
    ( "LEVELS:" "ENTER RECURSION LEVELS" 0 )
  )
  ( 2 2 )
  4 ROLL DUP INFORM
REPEAT
  DUP OBJ $\rightarrow$  DROP
  « @  $\leftarrow Dline$ 
  «
    65 ROT + R $\rightarrow$ B 32 ROT - R $\rightarrow$ B 2  $\rightarrow$ LIST
    ROT 65 + R $\rightarrow$ B 32 4 ROLL - R $\rightarrow$ B 2  $\rightarrow$ LIST
    LINE
  »
»

```

```

« @ ←Point
  → x1 y1 x2 y2
  «
    IF 'x2-x1==0'
    THEN
      IF 'y2>y1'
      THEN 90 ELSE 270
    END
    ELSE
      y2 y1 - x2 x1 - / ATAN
      57.2957795131 *
    END
    IF 'x1>x2' THEN 180 +
    END
  »
»
« @ ←Generate
  0 0 → x y height angle level x1 y1
  «
    x '←turtx' STO
    y '←turty' STO
    height '←turtr' STO
    '←turtx' ←turtt 1.74532925199E-2 *
    COS ←turtr * STO+
    '←turty' ←turtt 1.74532925199E-2 *
    SIN ←turtr * STO+
    ←turtx 'x1' STO
    ←turty 'y1' STO
    'level' 1 STO-
    x y x1 y1 ←Dline EVAL
    IF 'level>0'
    THEN
      x y x1 y1 ←Point EVAL '←turtt' STO
      '←turtt' ←lfang STO+
      ←turtx ←turty ←lhf
      height * ←lfang
      level ←Generate EVAL
      x y x1 y1 ←Point EVAL '←turtt' STO
      '←turtt' ←rtang NEG STO+
      x1 y1 ←lhf height * ←rtang
      level ←Generate EVAL
    END
  »
»

```

```

0 0 0 0 0 0 0 -30 0 0 RCLF
→
←lfa1p ←rtal1p ←lfang ←rtang
←height level
←Dline ←Point ←Generate
←turtt ←turtr ←turtx ←tury
←lhf ←rhf x y x1 y1 flags
«
-3 SF RAD
# 131d # 64d BLANK PICT STO
( # 0d # 0d ) PVIEW
'←lhf=2^(-2/(3*←lfa1p))' DEFINE
'←rhf=2^(-2/(3*←rtal1p))' DEFINE
x y x1 y ←height + ←Dline EVAL
x y x1 y ←height + ←Point EVAL
'←turtt' STO
'←turtt' ←lfang STO+
x1 y ←height + ←lhf ←height *
←lfang level ←Generate EVAL
x y x1 y ←height + ←Point EVAL
'←turtt' STO
'←turtt' ←rtang NEG STO+
x1 y ←height + ←rhf ←height *
←rtang level ←Generate EVAL
( ) PVIEW
flags STOF
»
» EVAL
END
»

```

Julia Sets

To display Julia sets the program JULIA calculates a non-attracting fixed point z_0 for a starting position, then calculates preimages of the function $f_c(z)=z^2+c$. The fixed point is calculated by first examining $|1 + \sqrt{1-4c}|$. If this value is greater than 1, then the fixed point is

$$z_0 = \frac{1 - \sqrt{1-4c}}{2} \quad \text{otherwise} \quad z_0 = \frac{1 + \sqrt{1-4c}}{2}.$$

The preimages of z_n are $\pm\sqrt{z_{n+1}-c}$. As the program iterates, the preimages are randomly selected, and the resulting image shows points on the Julia set. This calculation method has the advantage of being small and reasonably efficient, and so is well suited to the HP 48, but you may notice differences with pictures resulting from other methods.

JULIA 206 Bytes Checksum #B5B9h

n (x,y) (x_{min},y_{min}) (x_{max},y_{max}) →

```

«
PICT PURGE                                Purge existing PICT
{ #0 #0 } PVIEW                           Show PICT
PMAX PMIN                                 Scale PICT
→ n c
«
1 c 4 * - √ 1 +                            Calculate z0
DUP IF ABS 1 ≤
THEN 2 - NEG
END 2 /
1 n START                                Loop for n iterations
c - √                                    Calculate preimage
IF RAND .5 < THEN NEG END                Randomly use -preimage
DUP DUP PIXON                            Show new point and
NEG PIXON                                its mirror image
NEXT DROP
{ } PVIEW                                Show PICT, wait for CANCEL
»
»

```

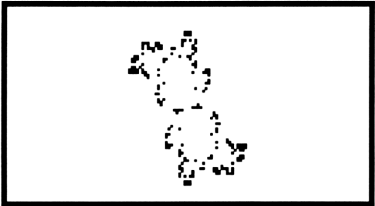
Idea: Modify this program to use INFORM.

Some examples:

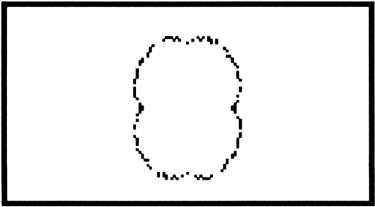
<i>n</i>	<i>c</i>	<i>PMIN</i>	<i>PMAX</i>
750	(-.75,0)	(-1.6,-1)	(1.5,1)



<i>n</i>	<i>c</i>	<i>PMIN</i>	<i>PMAX</i>
400	(.11,.66)	(-3.5,-1.25)	(3.5,1.25)



<i>n</i>	<i>c</i>	<i>PMIN</i>	<i>PMAX</i>
150	(-.2,0)	(-2.75,-1.25)	(2.75,1.25)

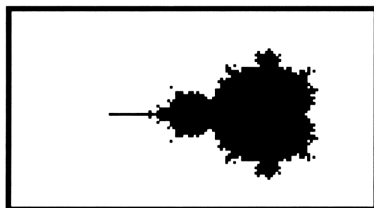


The Mandelbrot Set

The Mandelbrot set is the set of complex numbers c in the complex plane for which the sequence $z_{n+1}=z_n^2+c$ is bounded. The HP 48 is not well-suited to displaying the Mandelbrot set, but a crude approximation can be created with the program MANDEL. There are many algorithms for this problem. In MANDEL, the bound is calculated as

$$\frac{1 + \sqrt{1 + 4|c|}}{2}$$

and 15 iterations are taken to test whether the sequence remains within the calculated bound. Since the Mandelbrot set is symmetric around the x axis, the upper half of the plot is calculated and the points are reflected below the axis. To speed up the process, a line from $(-1.95,0)$ to $(-.12,.99)$ is used as the left boundary for iterates in x . The equation $x_{\text{left}}=(y-1.0655)/.5464$ defines the left border for each row of values to be calculated. The program can run for up to 20 minutes.



To “zoom in” on a small area requires more effort and is considerably slower, because you can’t take advantage of symmetry about an axis. A program that will do this is left as an exercise for the user and the HP 48’s batteries.

```

«
PICT PURGE                                Purge existing PICT
(-3,-1.2) (1,1.23) PDIM                 Scale PICT in user units
( #0d #0d ) PVIEW                         Display PICT
1 CF 0 .96                               Initialize signal flag; loop 0-.96
FOR y
  y 1.0655 - .5464 / .6                   Calculate range for loop
  FOR x
    x y R>C                               Generate c
    DUP ABS 4 * 1 + √                     Calculate bound
    1 + 2 /
    (0,0) 1 15                            z0
    FOR n                                  18 iterations
      IF DUP ABS 3 PICK >                 If bound exceeded
      THEN 1 SF 99 'n' STO                 Then set flag, terminate loop
      ELSE SQ 3 PICK +                     Else calculate next iterate
      END
    NEXT DROP2                             Discard iterate, bound
    IF 1 FC?C                             If bound was exceeded
    THEN
      DUP PIXON CONJ PIXON                Set pixels
      ELSE DROP
      END
    .03053 STEP
    .03796 STEP
  ( ) PVIEW
«
Show PICT, wait for CANCEL

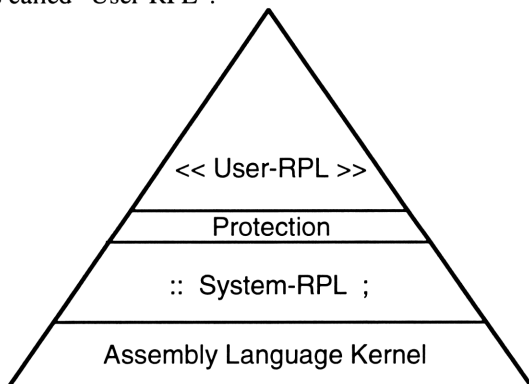
```

System Programming

This chapter provides a *limited* introduction to the potential benefits of using some of the underlying system resources in your programs. It is beyond the scope of this handbook to present a complete description of all the resources available. Additional documentation and internal development tools are provided by both Hewlett-Packard and other third-party vendors. The Hewlett-Packard bulletin board (listed in the HP 48 User's Guide) is one place to find further information.

User-RPL vs. System-RPL

The kernel of the HP 48 operating system/language known as RPL has been written in assembly language, and much of the functionality in the HP 48 is implemented in what is sometimes called "System-RPL". Programs entered from the keyboard of the HP 48 are written in what is sometimes called "User-RPL".



Programs written in User-RPL and System-RPL share the same resources, stack, etc. The commands available in User-RPL represent a subset of the functionality available in System-RPL.

From a practical point of view there are three main distinctions between User-RPL and System-RPL. First, User-RPL commands have names that are recognized when you enter them into the command line,

whereas System-RPL objects must be accessed via the SYSEVAL command. Second, User-RPL commands have extra code responsible for validating input arguments (and thus require a bit of extra execution time), whereas programs written in System-RPL have no error protection. This layer of protection insures that invalid input arguments do not result in undesirable behavior by underlying code. Finally, there are more resources available to System-RPL programs.

SYSEVAL & Version Identification

The command SYSEVAL is the “doorway” to System-RPL objects. This command accepts a memory address formed as a user binary integer and evaluates the object at that address. If the address is incorrect, the HP 48 will likely “hang up” or lose memory. As you experiment with System-RPL objects, you may wish to define user programs that perform common System-RPL tasks. This chapter includes an assortment of small user programs to help you along. Frequent backups of memory are also a good idea.

The HP 48G/GX command VERSION returns the version of the operating system and a copyright message. To identify the version of the operating system on any HP 48, use the command SYSEVAL to execute the internal object VERSTRING. To do this, place the user binary integer **#30794h** (the address of VERSTRING) on the stack and evaluate SYSEVAL. The result is a string in the form **"HHP48-*A*"**. The last letter represents the version of the operating system. The following program returns a string in level 2 containing the letter representing the operating system version, and 0 in level 1 if the calculator is a HP 48S/SX or a 1 if the calculator is a HP 48G/GX:

GETVER 54.5 Bytes Checksum #6452h

→ “version” T/F

```

«
  #30794h SYSEVAL      Get the version string
  8 8 SUB              Extract the last character
  DUP "J" >           Duplicate and compare to "J"
»

```

It is important to remember that SYSEVAL is a command like any other, and saves its last argument (if flag -55 is clear), *not* the arguments processed by the system object.

The Dangers & Benefits of System-RPL

The danger of supplying improper arguments to a System-RPL object can be gently illustrated with the multiplication function. In User-RPL executing the program `« 2 "JAMES" * »` results in the error Bad Argument Type. If you place the arguments 2 “JAMES” on the stack and execute `#2A9BCh SYSEVAL` (the System-RPL object `%*` for real number multiplication) you get an unexpected result: `-1.06910683E16`. In this instance the operating system remained intact – the multiply code actually interpreted the data located at the string “JAMES” as a number. The usual result for errors resulting from the improper application of System-RPL objects is the *total loss of memory*.

Applications written entirely in System-RPL can achieve greater performance with a wider set of resources at the risk of being completely responsible for error trapping. That level of implementation is beyond the scope of this chapter. However, some applications may benefit from a limited use of System-RPL objects. In this chapter a series of objects will be presented which may be viewed as an additional “toolbox” for your programming efforts.

Assumptions in This Chapter

There are two basic assumptions underlying the examples in this chapter. First, the HP 48 is in HEX mode (the base mode affects the display of internal binary integers and checksums), and second, you have backed up your HP 48.

Fixed Entry Points

There have been many versions of the operating system software for the HP 48 calculator since its original introduction in 1990. With the exception of the dramatic changes embodied in the HP 48G and

HP 48GX models, the operating system has remained fairly stable. In order to produce a program that is guaranteed to run on all models of the HP 48, the use of System-RPL objects should be restricted to those objects whose location has remained stable. Objects reside at a specific address in memory, often known as an *entry point*. It should be obvious that robust programs should be able to run on all versions of the HP 48. A program that works on only one version of the HP 48 operating system can generate a “Memory Lost” error when distributed to a wide audience, because there is no guarantee that all users will have the same version of the operating system. Hewlett-Packard’s list of fixed entry points represents the only known list of entries that have not changed since the first version of the HP 48 was introduced. The objects listed in this chapter are a small subset of that list, and are intended to expand your programming “toolbox”.

!! WARNING !!

The objects presented in this chapter *are not supported by Hewlett-Packard* and *must be used carefully*. Please do not contact either Hewlett-Packard or the author for further technical advice on the use of these objects or other aspects of System-RPL programming. Neither the author nor Hewlett-Packard take any responsibility for improper use of system commands that may result in loss of data or damage to your calculator.

Examples in this Chapter

The example programs in this chapter are small, and intended to be “building blocks” for larger applications. Some example programs call other example programs. The example programs COERCE and UNCOERCE are called most frequently. When an example program calls another example program in this book, the program name will be printed in bold. For instance, the program `3 + FRED 2 * *` calls the example program **FRED**. Each program is documented with the its size in bytes and a checksum value. When you enter the example program into your HP 48, use the command BYTES to verify that you

have entered the program correctly. BYTES returns the size of the program and a checksum, which should match the checksum printed in this book. The checksum comparison is especially useful for making sure you've entered addresses of System-RPL objects correctly.

Type Checking. Some example programs in this chapter contain extra type checking code that is not absolutely required *if you never make mistakes*. The program CHKARGS has been included to provide a general argument checking mechanism. This approach has been taken to let you experiment with system programming with some reduced risk. Experienced programmers may wish to remove the type checking to improve program performance.

Naming Conventions

Some System-RPL objects have names given by Hewlett-Packard that contain more than 8 characters. Example programs that use System-RPL objects have been given names that reflect the System-RPL object being demonstrated where possible. Sample program names have been chosen to make the programs distinct within the HP 48 VAR menu and readable in a DOS directory.

Hewlett-Packard uses special symbols to distinguish object types in names and stack diagrams. Some common symbols are listed below:

Symbol	Object	Type
#	Internal binary integer	20
%	Real number	0
%%	Extended real	21
C%	Complex number	1
C%%	Extended complex	22
\$	String	2
chr	Character object	24
hxs	User binary integer	10
id	Name	6
{ }	List	5
comp	Composite object (list, program, unit, etc.)	5, 8, 9, 13
::	Program	8

Checking Arguments

The protection code for User-RPL command and functions validates the number and type of input arguments. If an application is going to accept input from an inexperienced user, it may be safer to check the arguments before they're supplied to unprotected System-RPL objects.

The program CHKARGS takes a list in level one that contains type numbers for each required stack argument. The first number in the list specifies the desired type for the object in stack level 1, the second number specifies the desired object type for level 2, and so on. For instance, if a string is required in level 2 and a real number is required in level 1, the input to CHKARGS would be the list { 0 2 }.

For a list of object types and their type numbers, see *Object Types*.

CHKARGS 151.5 Bytes Checksum 3981h

obj_n ... obj₁ { type₁ ... type_n } →

```
«
  DUP SIZE → types n
  «
    IF DEPTH n <                               Verify # of args
    THEN 513 DOERR
    END
    1 n FOR i                                     Loop for each argument
      i PICK TYPE types i GET                   Get ob type, req'd ob type
      IF ≠ THEN 514 DOERR END                   Error out if not the same
    NEXT
  »
»
```


Binary Integers

There are a number of object types used by System-RPL objects which are not available in the User-RPL environment. The most prevalent of these is the *internal binary integer*. Internal binary integers (sometimes nicknamed *bints*) are unsigned 20-bit quantities that are useful for many functions. These integers differ from user binary integers, which are actually stored internally as hex strings. To avoid confusion, this chapter will use the terms *user binary integer* and *internal binary integer*. While user binary integers (object type 10) are displayed with a leading # character, internal binary integers (object type 20) are displayed within <> symbols. A trailing character indicates the base display mode. For instance, if the base mode of the HP 48 is binary, then the internal binary integer 5 would be displayed as <101b>.

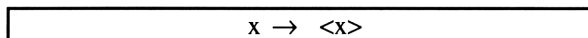
Internal binary integers live in the range $0 \leq n \leq \text{FFFFF}$. If you subtract <1h> from <0h>, you get <FFFFFh> (decimal 1048575). No overflow or underflow indications are available.

Type Conversions. Internal binary integers cannot be entered directly into the HP 48, so conversion routines are needed. There are a number of useful System-RPL objects which convert between real numbers and internal binary integers. The most basic of these are COERCE and UNCOERCE. COERCE converts a real number to an internal binary integer, and UNCOERCE converts an internal binary integer into a real number.

The HP 48 will not recognize the names COERCE and UNCOERCE – these are names that Hewlett-Packard gives the routines – so you must refer to these objects by address. The address of COERCE is #18CEAh, and the address of UNCOERCE is #18DBFh.

To begin to experiment with internal binary integers, use the following two programs, which we'll name after the internal objects that are used.

COERCE 54 Bytes Checksum #60F4h

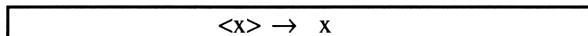


```

«
{ 0 } CHKARGS      Confirm real number as input
#18CEAh SYSEVAL    Convert to internal binary integer
»

```

UNCOERCE 64 Bytes Checksum #52Ah



```

«
{ 20 } CHKARGS      Confirm internal binary integer as input
#18DBFh SYSEVAL     Convert to real number
»

```

A test has been added in each of these programs to be sure that the proper object is being converted. You can remove the test later on when you're absolutely certain that you'll never make a mistake.

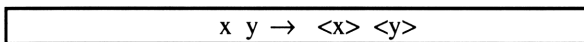
Now use the program COERCE to convert the real numbers -1, 0, 4.49, 4.5, 5, and 6.75E37 to internal binary integers. Assuming that the HP 48 is in HEX mode, you should get the following results:

Input	Result
-1	<0h>
0	<0h>
4.49	<4h>
4.5	<5h>
5	<5h>
6.75E37	<FFFFFFh>

Notice that values less than 0 convert to <0h>, values greater than 1048575 convert to <FFFFh>, fractional parts < .5 round to the next lowest integer, and that fractional parts ≥ .5 round to the next highest integer.

Some System-RPL objects require two internal binary integers as parameters. The System-RPL object COERCE2 (address #194F7h) is useful for converting two real numbers into internal binary integers. The program COERCE2 uses this object after checking the arguments.

CRCE2 55.5 Bytes Checksum #F685h



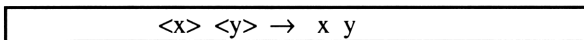
```

⌞
{ 0 0 } CHKARGS           Both real numbers?
#194F7h SYSEVAL           Convert 2 reals to bints
⌘

```

The corollary to COERCE2 is UNCOERCE2 (address #1950Bh), which converts two internal binary integers into real numbers. The following program UCRC2 uses this object after checking the arguments.

UCRC2 71.5 Bytes Checksum #C6Fh



```

⌞
{ 20 20 } CHKARGS        Both bints?
#1950Bh SYSEVAL          Convert 2 bints to reals
⌘

```

Internal Binary Integer Operations. The following System-RPL objects operate on a single internal binary integer (bint):

Command	Address	Description
#1+	#03DEFh	Adds <1d> to a bint
#1-	#03E0Eh	Subtracts <1d> from a bint
#2+	#03E2Dh	Adds <2d> to a bint
#2-	#03E4Eh	Subtracts <2d> from a bint
#2*	#03E6Fh	Multiplies a bint by <2d>
#2/	#03E8Eh	Returns FLOOR(bint/2d)
#3+	#6256Ah	Adds <3d> to a bint
#3-	#625FAh	Subtracts <3d> from a bint
#4+	#6257Ah	Adds <4d> to a bint
#4-	#6260Ah	Subtracts <4d> from a bint
#5+	#6258Ah	Adds <5d> to a bint
#5-	#6261Ah	Subtracts <5d> from a bint
#8+	#625BAh	Adds <8d> to a bint
#8*	#62674h	Multiplies a bint by <8d>
#10+	#625DAh	Adds <10d> to a bint
#10*	#6264Eh	Multiplies a bint by <10d>
#12+	#625EAh	Adds <12d> to a bint

The following System-RPL objects operate on two internal binary integers:

Command	Address	Description
#*	#03EC2h	Multiplication: returns <bint>
#+	#03DBCh	Addition: returns <bint>
#-	#03DE0h	Subtraction: returns <bint>
#/	#03EF7h	Division: returns <remainder> (level 2) and <quotient> (level 1)
#MAX	#624C6h	Returns <maximum of two bints>
#MIN	#624BAh	Returns <minimum of two bints>

Internal Binary Integer Constants. The following objects place bints on the stack:

Command	Address	Stack Output
MINUSONE	#6509Eh	<FFFFFh>
ZERO	#03FEFh	<0d>
ONE	#03FF9h	<1d>
TWO	#04003h	<2d>
THREE	#0400Dh	<3d>
FOUR	#04017h	<4d>
FIVE	#04021h	<5d>
SIX	#0402Bh	<6d>
SEVEN	#04035h	<7d>
EIGHT	#0403Fh	<8d>
NINE	#04049h	<9d>
TEN	#04053h	<10d>
ELEVEN	#0405Dh	<11d>
TWELVE	#04067h	<12d>
THIRTEEN	#04071h	<13d>
FOURTEEN	#0407Bh	<14d>
FIFTEEN	#04085h	<15d>
SIXTEEN	#0408Fh	<16d>
SEVENTEEN	#04099h	<17d>
EIGHTEEN	#040A3h	<18d>
NINETEEN	#040ADh	<19d>
TWENTY	#040B7h	<20d>
THIRTYTWO	#0412Fh	<32d>
FORTYEIGHT	#64B3Ah	<48d>
FIFTYFIVE	#64B80h	<55d>
FIFTYSIX	#64B8Ah	<56d>
FIFTYSEVEN	#64B94h	<57d>
SIXTYTWO	#64BC6h	<62d>
SIXTYTHREE	#64BD0h	<63d>
SIXTYFOUR	#64BDAh	<64d>
BINT_131d	#64D24h	<131d>
BINT255d	#64E28h	<255d>
ZEROZERO	#641FCh	<0> <0>
ONEONE	#63AC4h	<1> <1>

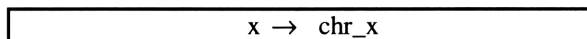
Character Constants

A *character constant* is an internal object type that represents a single character. A character object is a distinct object type (object type 24), *not* a one character string. The HP 48 displays the word **Character** when a character object is on the stack. The following System-RPL objects are available for conversion between character and other object types:

Command	Address	Description
#>CHR	#05A75h	Converts a bint to a character
CHR>#	#05A51h	Converts a character to a bint
CHR>\$	#6475Ch	Converts a character object to a 1 character string object

The program TOCHR creates a character object specified by a real number. TOCHR calls the program COERCE to create an internal binary integer, then uses the object #>CHR to create the character.

TOCHR 44.5 Bytes Checksum #3BFEh



«

COERCE

Convert the real number to a bint

#5A75h SYSEVAL

Convert the bint to a character object

»

The #>CHR conversion uses only the lower 8 bits of the 20 bit binary integer argument. The lower 8 bits of 999 are 11100111, so if you supply 999 as the input argument to TOCHR, you get the character whose number is 231 decimal. This conversion is most useful when you already have an internal binary integer on the stack – otherwise the User-RPL command CHR works just fine.

Character Strings

A number of System-RPL objects are available for manipulating character strings (object type 2). There is no error checking for these objects, so it may be possible to specify an operation that extends beyond the boundary of a string, or returns a meaningless result.

Converting Between Name Objects and Strings. The objects ID>\$ and \$>ID may be used to convert between a name object (object type 6, internally known as an identifier, or an id) and a string. For example, the sequence « "FRED" #5B15h SYSEVAL » returns the name object 'FRED'.

\$>ID	#05B15h
Converts string to name object "string" → 'name'	
ID>\$	#05BE9h
Converts name object to a string 'name' → "string"	

Converting Objects to Strings. The User-RPL command →STR converts a real number to a string in the current display format. If you want to convert the number to a string using the STD or stack display formats, the objects EDITDECOMP\$ and a%>\$, are useful.

EDITDECOMP\$	#15B13h
Converts an object to a string in a format suitable for editing in the command line. object → "string"	
a%>\$,	#162ACh
Converts a real number to a string (with commas if format is FIX) % → "string"	

Adding Character Objects to a String. The objects >H\$ and >T\$ may be used to add a character object to the head (>H\$) or tail (>T\$) of a string.

>H\$	#0525Bh
Adds a character object to the head of a string “string” chr → “string”	
>T\$	#052EEh
Adds a character object to the tail of a string “string” chr → “string”	

Given a string in level 2 and real number that specifies a character in level 1, the User-RPL program to add the character to the string would be `« CHR + »`. This same task can be performed with the System-RPL objects COERCE, #>CHR, and >T\$ (proving that User-RPL is sometimes the more efficient tool):

ADDCHR 87.5 Bytes Checksum #E953h

“string” x → “stringchr_x”

```
«
  { 0 2 } CHKARGS  Confirm string and real number
  #18CEAh SYSEVAL  Convert the real number to a bint
  #5A75h SYSEVAL   Convert the bint to a character object
  #52EEh SYSEVAL   Add the character to the end of the string
»
```

Creating a String of Blank Characters. The object Blank\$ may be used to create a string of space (character code 32) characters.

Blank\$	#45676h
Creates a string of <i>n</i> spaces <i><n></i> → “string”	

The program SPACES takes as input a real number in level 1 and returns a string containing the specified number of spaces:

SPACES 45.5 Bytes Checksum #36DFh

$x \rightarrow \text{"string"}$

⌘

COERCE *Convert the real number to a bint*
#45676h SYSEVAL *Create the string*

⌘

Retrieving Messages. Messages can be retrieved from the built-in message table with the object JstGETTHEMSG. If a localized message table has been installed in the HP 48 the localized message will be returned. Messages built into the HP 48 are listed in *Messages*.

JstGETTHEMSG	#04D87h
Returns a message from the message table	
$\langle \text{bint} \rangle \rightarrow \text{"string"}$	

The program GETMSG takes as input a real number and returns the corresponding message:

GETMSG 45.5 Bytes Checksum #DB2Fh

$x \rightarrow \text{"string"}$

⌘

COERCE *Convert the real number to a bint*
#4D87h SYSEVAL *Get the message string*

⌘

Example: 3095 GETMSG returns the string "Invalid Name".

Searching Strings. The objects POS\$ and POS\$REV provide more control over string searches than is available with the User-RPL command POS.

POS\$	#645B1h
Searches for <i>\$find</i> in <i>\$search</i> starting at <i>pos</i> ($1 \leq pos \leq \text{string length}$). Returns <0> for unsuccessful search.	
<i>\$search</i> <i>\$find</i> <pos> → <pos> or <0>	
POS\$REV	#645BDh
Searches for <i>\$find</i> in <i>\$search</i> starting at <i>pos</i> ($1 \leq pos \leq \text{string length}$) and searching backwards. Returns <0> for unsuccessful search.	
<i>\$search</i> <i>\$find</i> <pos> → <pos> or <0>	

The program NXTSTR uses the object POS\$ to look for the instance of a find string (in level 2) within a search string in level 3. The starting position is given in level 1. If the search is unsuccessful, 0 is returned.

NXTSTR 90 Bytes Checksum #941Ch

"search" "find" start → position

```

«
{ 0 2 2 } CHKARGS      Check arguments
#18CEAh SYSEVAL        Convert the starting position to a bint
#645B1h SYSEVAL        Perform the search
#18DBFh SYSEVAL        Convert the result into a real number
»

```

Example: The sequence « "12341256127" "12" 3 NXTSTR » EVAL returns the result 5.

To illustrate reverse string searching, the program LSTSTR uses the object POS\$REV. The arguments are the search string in level 2 and the string to find in level 1. The position is returned as a real number, which is 0 if the find string is not found.

LSTSTR 92.5 Bytes Checksum #7563h

“search” “find” → position

```

«
{ 2 2 } CHKARGS      Check arguments
OVER SIZE
#18CEAh SYSEVAL      Length of the search string as a bint
#645BDh SYSEVAL      Perform the search
#18DBFh SYSEVAL      Convert the result into a real number
»

```

Example: The sequence « "123412561278" "12" LSTSTR » EVAL returns the result 9.

Finding Character Codes. The object SUB\$1# is useful for finding the character code of an arbitrary character in a string. In User-RPL, a program to do this given a string in level 2 and a real number in level 1 would be « DUP SUB NUM ». If the result needs to be a bint, it’s much more efficient to use SUB\$1#, which is fast and avoids the creation of a substring in temporary memory.

SUB\$1#	#30805h
Returns the bint value of the specified character in a string	
“string” <pos> → <value>	

Parsing a String. The object palparse performs the same parsing action as the **[ENTER]** key. This object returns either the object represented by the string and the internal flag TRUE, or the string, position of the error, remainder of the string, and the internal flag FALSE.

palparse #238A4h
 Parses *string* into an object. Returns TRUE if the string represents a valid object, or the position of the error, remaining string, and FALSE.
 “string” → object TRUE
 or: “string” → “string” <pos> string’ FALSE

The object palparse is useful for instances when a string needs to be parsed and the output should not be evaluated. For instance, you might want to place the command + on the stack. Parsing the string “+” is one way to accomplish this. The program PARSE simplifies the use of palparse. A string is supplied, and the result in level 1 is 0 if the parsing was unsuccessful, or 1 if the string was valid. If the result is 1, the object is returned in level 2. This program uses the System-RPL object COERCEFLAG* (address #5380E), which is described in *Tests*.

PARSE 108.5 Bytes Checksum #CCBCh

“string” → object 1 *Successful*
 “string” → 0 *Failed*

```

«
{ 2 } CHKARGS                               Verify string input
#238A4 SYSEVAL                               Attempt to parse the string
IF « #5380E SYSEVAL » EVAL                  Convert the flag to 0 or 1
THEN 1                                       Return ob and 1 if OK
ELSE 3 DROPN 0                              Else return 0
END
»

```

* An extra program shell is required around COERCEFLAG to make the program work, since COERCEFLAG does an implied ».

Real and Extended Real Numbers

Internal calculations involving real numbers (object type 0) are performed using *extended real* numbers (object type 21) to maximize the precision of calculations. A real number consists of a sign, 12 digit mantissa, and 3 digit exponent. An extended real number consists of a sign, 15 digit mantissa, and 5 digit exponent. Exponents are stored in 10s complement form. Real exponents live in the domain $-500 < \text{EEE} < 500$, and extended real exponents live in the domain $-50000 < \text{EEEE} < 50000$.

When an extended real number is on the stack, the HP 48 displays the words **Long Real**.

The naming convention for System-RPL objects uses % to denote a real number and %% to denote an extended real number. Object descriptions in this chapter use the same notation.

Real Number Conversions. Sometimes it's desirable to maintain intermediate results of a long calculation as extended reals to minimize roundoff errors. The following System-RPL objects are used to convert between real and extended real numbers:

%>%% Converts a real number to an extended real number $\% \rightarrow \%\%$	#2A5C1h
%%>% Converts an extended real number to a real number $\%\% \rightarrow \%$	#2A5B0h
2%>%% Converts two real numbers to extended real numbers $\% \ \% \rightarrow \%\% \ \%\%$	#2B45Ch
2%%>% Converts two extended real numbers to real numbers $\%\% \ \%\% \rightarrow \% \ \%$	#2B470h

The following two programs are useful for frequent conversions between real numbers and extended real numbers. RTOER converts a real number to an extended real number; ERTOR performs the reverse conversion.

RTOER 53 Bytes Checksum #2D11h

$\% \rightarrow \%\%$

```

«
{ 0 } CHKARGS           Is this a real number?
#2A5C1h SYSEVAL         Convert to extended real number
»

```

ERTOR 61 Bytes Checksum #E144h

$\%\% \rightarrow \%$

```

«
{ 21 } CHKARGS          Is this an extended real number?
#2A5B0h SYSEVAL         Convert to real number
»

```

Some System-RPL objects require two extended real numbers as parameters. The System-RPL command $2\%>\%\%$ (address #2B45Ch) is useful for converting two real numbers into extended real numbers. The program RTOE2 uses this command after checking the arguments.

RTOE2 55.5 Bytes Checksum #7783h

$\% \ \% \rightarrow \%\% \ \%\%$

```

«
{ 0 0 } CHKARGS         Both real numbers?
#2B45Ch SYSEVAL         Convert 2 reals to extended
                        real numbers
»

```

The corollary to 2%>%% is 2%%>% (address #2B470h), which converts two extended real numbers into real numbers. The program ETOR2 uses this object after checking the arguments.

ETOR2 71.5 Bytes Checksum #A5BDh

%% %% → % %

⌘ { 21 21 } CHKARGS *Both extended reals?*
 #2B470h SYSEVAL *Convert 2 extended real*
 ⌘ *numbers to reals*

Extended Real Functions. The following objects operate on extended real numbers:

%%+	#2A943h
Adds two extended real numbers	
%% %% → %%	
%%-	#2A94Fh
Subtracts two extended real numbers	
%% %% → %%	
%%*	#2A99Ah
Multiplies two extended real numbers	
%% %% → %%	
%%/	#2A9E8h
Divides two extended real numbers	
%% %% → %%	
%%^	#2AA5Fh
Raises extended real in level 2 to power of extended real in level 1	
%% %% → %%	
%%ABS	#2A8F0h
Absolute value of an extended real number	
%% → %%	
%%ANGLE	#2AD4Fh
Angle from rectangular coordinates	
%%x %%y → %%angle	

%%CHS Change sign	## → ##	#2A910h
%%COS Cosine	## → ##	#2AC57h
%%COSH Hyperbolic cosine	## → ##	#2ADC7h
%%EXP e^x	## → ##	#2AB1Ch
%%FLOOR Returns greatest integer $\leq x$	## → ##	#2AF99h
%%H>HMS Converts decimal hours to HH.MMSS form	## → ##	#2AF27h
%%LN Natural log	## → ##	#2AB5Bh
%%LNP1 $\ln(x+1)$	## → ##	#2AB94h
%%MAX Returns larger of two numbers	## ## → ##	#2A6DCh
%%SIN Sine	## → ##	#2AC06h
%%SINH Hyperbolic sine	## → ##	#2AD95h

%%SQRT Square root	#2AAEAh
%% → %%	
%%TANRAD Tangent using radians	#2ACA8h
%% → %%	

Rectangular/Polar Conversions. The following objects convert between rectangular and polar coordinates using the current angle mode:

%%REC>%POL Real number rectangular to polar conversion	#2B48Eh
%x %y → %radius %angle	
%%POL>%REC Real number polar to rectangular conversion	#2B4BBh
%radius %angle → %x %y	
%%R>P Extended real number rectangular to polar conversion	#2B498h
%%x %%y → %%radius %%angle	
%%P>R Extended real number polar to rectangular conversion	#2B4C5h
%%radius %%angle → %%x %%y	

Example: Log to an Arbitrary Base. The common logarithm (LOG) and natural logarithm (LN) functions are built into the HP 48, but logs to other bases must be calculated, and the process often introduces small roundoff errors. To calculate a log to the base a , use the rule $\text{LOG}_a b = \text{LOG}(b)/\text{LOG}(a)$.

To demonstrate the the effect of roundoff errors, calculate 2^{34} , which yields 17,179,869,184. Suppose you wish to reverse the process and ask “What power of 2 is this?” Apply the rule above using both the LOG and LN functions. Press: **[ENTER][LOG][2][LOG][÷][SWAP][LN][2][LN][÷]**. The answer should be 34, but notice that neither LOG or LN got the right answer.

The program LOGB performs the calculation using extended real numbers. LOGB requires the number in level two and the base in level 1.

LOGB 87 Bytes Checksum #60C3h

$x \ n \rightarrow \log_n(x)$

⌘ RT0E2 #2AB5Bh SYSEVAL SWAP #2AB5Bh SYSEVAL SWAP #2A9E8h SYSEVAL ERTOR ⌘	<i>Convert reals to extended reals</i> <i>Take the log of the base</i> <i>Take the log of the number</i> <i>Divide</i> <i>Convert result to real number</i>
-----------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Displaying Extended Reals. When an extended real number is on the stack, the HP 48 displays the words **Long Real**. If you wish to examine the digits in the extended real number, you need a tool to perform a conversion. Several small utility programs are presented here to facilitate the use and display of extended real numbers:

MKERTOA	Creates the program ERTOA.
ERTOA	Converts an extended real number object to a string object showing all the digits.
DSPER	Displays stack levels 1 and 2 in the status area of the display using the small graphics font.
ERCALC	A custom menu for extended real calculations.

The program ERTOA may be used to display the digits in an extended real number. The technique used is to execute an assembly language code object that reads the data in the body of an extended real object and writes the appropriate characters to the body of a new string object.

Since ERTOA cannot be entered from the keyboard the program MKERTOA (shown on the next page) is used to create ERTOA. Enter the program MKERTOA, verify its checksum, then execute it to create ERTOA. Once ERTOA has been created, you can use it to see all the digits in an extended real number.

Example: Display the extended real square root of 3:

```
« 3 RTOER #2AAEAh SYSEVAL ERTOA »  
→  
" 1.73205080756887E+00000"
```

Notice that the string is too wide to fit completely in the display. The program DSPER (listed on the page after MKERTOA) can be used to display the contents of stack levels 1 and 2 as graphics objects on the stack display. By using the small graphics font, DSPER can easily display an extended real number.

MKERTO A uses the System-RPL objects CHANGETYPE, INNERCOMP and ::N to build the program. These are described in *Composite Objects*.

WARNING

Please verify the checksum of MKERTO A before you use it. If any of the numbers are incorrect, the code object that gets created will be incorrect, and will corrupt memory when used.

MKERTO A 1619.5 Bytes Checksum #EE15h

```

«
200 8 70 18 65 6 125 47 142 165 246 160 24 23
29 65 150 166 138 46 167 234 4 19 65 132 24 65
132 24 64 143 80 23 29 65 45 19 2 20 81 113 212
16 32 214 137 66 80 2 180 48 17 23 29 65 69 19
234 95 106 23 29 65 150 166 138 46 160 24 126
160 209 51 113 61 81 150 166 138 46 0 51 94 163
1 48 24 23 29 65 45 19 6 128 146 1 48 24 114 81
70 16 1 36 25 113 19 17 65 70 20 49 116 16 103
210 248 68 16 0 53 67 70 20 65 2 162 196 48 49
35 16 57 190 248 0 3 164 48 103 155 248 "" 1 143
START SWAP CHR + NEXT      Loop to accumulate bytes
11724 #18CEAh SYSEVAL      Create code object prologue
#5AB3h SYSEVAL              Execute CHANGETYPE
«                             Program shell
  IF DUP TYPE 21 ≠          Confirms object is extended real
  THEN 514 DOERR             Invalid Argument Type if not
  END
»
#54AFh SYSEVAL               Execute INNERCOMP
12 ROLL 3 ROLLD              Move code object into position
#3DEFh SYSEVAL               Execute #1+
#5445h SYSEVAL               Execute ::N to build program
'ERTO A' STO                  Store program in ERTOA
»

```

Once MKERTO A has been executed, it may be purged to save memory.

DSPER uses the System-RPL objects BlankDA1, XYGROBDISP and the example program XYGD, which are described further in this chapter under the heading *Graphics Operations*.

DSPER 375.5 Bytes Checksum #43D4h

```

«
  IF DEPTH
    THEN
      IF DUP DUP TYPE 21 ==
        THEN ERTOA
        ELSE →STR
        END
      IF DEPTH 2 >
        THEN
          3 PICK
          IF DUP TYPE 21 ==
            THEN ERTOA
            ELSE →STR
            END
          ELSE ""
          END
        ELSE
          "" ""
        END
      #3A546h SYSEVAL
      "2: " SWAP +
      1 →GROB
      { #0d #0d } { #131d #6d }
      SUB
      0 0 XYGD
      "1: " SWAP +
      1 →GROB
      { #0d #0d } { #131d #6d }
      SUB
      0 8 XYGD
      1 FREEZE
    »

```

Check for empty stack

*If level 1 ob is ext real
use ERTOA
otherwise convert to string*

Check for level 2 object

*If exists, get it
If level 2 ob is ext real
use ERTOA
otherwise convert to string*

Null \$ if no level 2 ob

*Display 2 null strings if
the stack is empty
Clear status display lines
Add level number to string
Make grob with small font*

*Trim to display width
Display level 2 grob
Add level number to string
Make grob with small font*

*Trim to display width
Display level 1 grob
Freeze display area 1*

An Extended Real Calculating Environment. The program ERCALC combines the tools described so far into a small extended real calculating environment by creating a custom menu.

ERCALC 777 Bytes Checksum #72E3h

```
«
{
  { "R→ER" « RTOER DSPER » }
  { "ER→R" « ERTOR DSPER » }
  { "%%+"
    « ( 21 21 ) CHKARGS #2A943h SYSEVAL DSPER »
  }
  { "%%- "
    « ( 21 21 ) CHKARGS #2A94Fh SYSEVAL DSPER »
  }
  { "%%*"
    « ( 21 21 ) CHKARGS #2A99Ah SYSEVAL DSPER »
  }
  { "%%/ "
    « ( 21 21 ) CHKARGS #2A9E8h SYSEVAL DSPER »
  }
  { "%%^ "
    « ( 21 21 ) CHKARGS #2AA5Fh SYSEVAL DSPER »
  }
  { "%%EXP"
    « ( 21 ) CHKARGS #2AB1Ch SYSEVAL DSPER »
  }
  { "%%LN"
    « ( 21 ) CHKARGS #2AB5Bh SYSEVAL DSPER »
  }
  { "%%J"
    « ( 21 ) CHKARGS #2AAEAh SYSEVAL DSPER »
  }
  DSPER
}
TMENU DSPER
»
```

ERCALC creates a temporary menu and executes DSPER. Each key performs its labeled function and executes DSPER to show the results.

Now you can use ERCALC to see what *actually* happened when you calculated the log base 2 of 17179869184 (2^{34}).

Execute ERCALC to display the temporary menu, then enter 2 34 $\boxed{y^x}$
R \rightarrow ER $\boxed{\text{NXT}}$ $\boxed{\text{2}}$ $\boxed{\text{NXT}}$ R \rightarrow ER $\boxed{\text{NXT}}$ $\boxed{\text{LN}}$:

```
2: 2.35670041390381E+00001
1: 6.93147180559945E-00001
4:
3:
2: Long Real
1: Long Real
 $\boxed{\text{2ndF}}$   $\boxed{\text{2ndF}}$   $\boxed{\text{LN}}$   $\boxed{\text{2ndF}}$   $\boxed{\text{DSPER}}$ 
```

Now press $\boxed{\text{NXT}}$ $\boxed{\text{2ndF}}$ to do the divide operation:

```
2:
1: 3.99999999999999E+00001
4:
3:
2:
1: Long Real
R $\rightarrow$ ER ER $\rightarrow$ R  $\boxed{\text{2ndF}}$   $\boxed{\text{2ndF}}$   $\boxed{\text{2ndF}}$   $\boxed{\text{2ndF}}$ 
```

Notice that the result is still not exactly 34. The rounding process that occurs during the conversion from an extended real number to a real number has the effect of rounding up to 34.

Composite Objects

The term *composite object* refers to any object built up of a series of *arbitrarily* typed objects. Lists and programs are examples of composite objects. (*Note:* an array is *not* a composite object.) While many operations on composite objects may be performed with the provided User-RPL commands, some additional System-RPL objects are useful to have available. In the stack diagrams below, a composite object is referred to as a *comp*.

The Size of a Composite Object. The objects LENCOMP and DUPLCOMP returns the number of objects in a composite object as an internal binary integer:

LENCOMP	#0567Bh
Returns the number of objects in a composite object	
comp → <n>	
DUPLCOMP	#63231h
Returns a composite and the number of objects in the composite	
comp → comp <n>	

First and Remaining Elements of a List. The objects CARCOMP and CDRCOMP return the first object in a composite or the composite object less its first element:

CARCOMP	#05089h
Returns the first object in a composite	
comp → ob ₁	
CDRCOMP	#05153h
Returns a composite less its first object	
comp → comp'	

If a composite object is null, CARCOMP returns a null composite of the same type. If a composite object is null or has one element, CDRCOMP returns a null composite of the same type.

The object NTHELCOMP returns the n th object within a composite and the flag TRUE or the flag FALSE if n is out of range (see *Tests* for a description of the flags TRUE and FALSE). If you're *absolutely certain* that n specifies an object within the composite, you can use the object NTHCOMPDROP. If you specify n out of range for NTHCOMPDROP, the HP 48 will surely wander "off the range" with possibly dire consequences (as has happened to the author on occasion).

NTHELCOMP	#056B6h
Returns the n th object in a composite and TRUE or FALSE comp $\langle n \rangle \rightarrow ob_n$ TRUE or FALSE	
NTHCOMPDROP	#62B9Ch
Returns the n th object in a composite comp $\langle n \rangle \rightarrow ob$	

Decomposing Composite Objects. The object INNERCOMP may be used to perform the equivalent of OBJ \rightarrow for composite objects.

INNERCOMP	#054AFh
Decomposes a composite object comp $\rightarrow ob_1 \dots ob_n \langle n \rangle$	
<i>Examples:</i>	
{ $ob_1 \dots ob_n$ } $\rightarrow ob_1 \dots ob_n \langle n \rangle$	
'A+B*C' \rightarrow 'A' 'B' 'C' * + $\langle 5h \rangle$	
⌘ 2 + ⌘ \rightarrow ⌘ 2 + ⌘ $\langle 4h \rangle$	

When a list is decomposed with INNERCOMP, the results are the same that you would expect from OBJ \rightarrow , except that the number of objects in the list is returned as an internal binary integer. A list can be built with the command { }N.

When a symbolic object is decomposed the objects are placed on the stack in the order in which they would be evaluated. A symbolic object can be built with the command SYMBN.

INNERCOMP can be quite interesting when applied to a program object, as shown in the stack diagram above. Notice that the program delimiters \llcorner and \lrcorner are actually returned as commands. These commands *must* be included at the appropriate ends of a User-RPL program. A program object may be built with the object ::N.

Extracting a Subset of a Composite. The object SUBCOMP may be used to extract a portion of a composite object, much the same way that the User-RPL command SUB works.

SUBCOMP	#05821h
Returns sub-composite.	
comp <start> <end> → comp	

Building Composite Objects. The objects {}N, SYMBN, and ::N all perform a similar operation – building a composite object from a series of objects on the stack.

{ }N	#05459h
Creates a list object from <n> objects on the stack ob ₁ ... ob _n <n> → { ob ₁ ... ob _n }	
SYMBN	#0546Dh
Creates a symbolic object from <n> objects on the stack ob ₁ ... ob _n <n> → 'symbolic'	
::N	#05445h
Creates a program object from <n> objects on the stack ob ₁ ... ob _n <n> → :: ob ₁ ... ob _n ;	

Building a program object is a special case, since ::N actually builds an internal version of a program, sometimes called a *secondary*. When a secondary is displayed on the stack, it merely looks like a program without the program delimiters. Program delimiters are essential, however, and should be included when you build a program object. Note that the \lrcorner (x>>, address #23639h) at the end of a program is different from the \lrcorner (x>>ABND, address #235FEh) used at the end of a program object that stores local variables.

Example:

```

«
  → x
«
  ....
  »      ← x>>ABND    (#235FEh)
  »      ← x>>        (#23639h)

```

Some “shortcut” objects are available for building small lists:

NULL{ }	#055E9h
Returns a null (empty) list	→ { }
ONE{ }N	#23EEDh
Creates a list containing one object	ob → { ob }
TWO{ }N	#631B9h
Creates a list containing two objects	ob ₂ ob ₁ → { ob ₂ ob ₁ }
THREE{ }N	#631CDh
Creates a list containing three objects	ob ₃ ob ₂ ob ₁ → { ob ₃ ob ₂ ob ₁ }

Composite Object Utilities.

EQUALPOSCOMP	#644A3h
Returns index of first object in a composite that match <i>ob</i> . Returns <0> if the object is not found.	composite ob → <pos>
Embedded?	#64127h
Returns TRUE if object in level 1 is equal to or embedded in the level 2 object, otherwise returns FALSE	ob ob → <i>flag</i>
apndvarlst	#35491h
Adds an object to a list if ob is not already in the list	{ list } ob → { list }

Tests

System-RPL objects which need to return “true” or “false” results do so with the system flag objects TRUE and FALSE (object type 27):

TRUE The system TRUE flag → TRUE	#03A81h
FALSE The system FALSE flag → FALSE	#03AC0h

Stack diagrams in this chapter use the term FLAG to represent a system flag that can be either TRUE or FALSE.

Flag Conversions. The object COERCEFLAG may be used to convert the internal flags into real numbers: TRUE maps to 1 and FALSE maps to 0.

COERCEFLAG Converts a system flag to a real number FLAG → %	#5380Eh
-------------------------------------------------------------------	---------

COERCEFLAG is useful, but performs an end-of-program action when finished. This means you must include the call to COERCEFLAG in an extra set of program delimiters:

« #5380E SYSEVAL » EVAL *Convert the flag to 0 or 1*

The program PARSE (earlier in this chapter) illustrates a use of COERCEFLAG.

To convert a real number flag into a system flag, use the object %0<>:

%0<> Returns TRUE if a real number is non-zero % → FLAG	#2A7CFh
---------------------------------------------------------------	---------

Flag Utilities. The following objects are available for manipulating flags:

AND Logical AND	#03B46h
FLAG FLAG → FLAG	
OR Logical OR	#03B75h
FLAG FLAG → FLAG	
NOT Logical NOT	#03AF2h
FLAG → FLAG	
XOR Logical XOR	#03ADAh
FLAG FLAG → FLAG	

Binary Integer Tests. The following objects compare two internal binary integers:

#= Equality	#03D19h
<bint> <bint> → FLAG	
#<> Inequality	#03D4Eh
<bint> <bint> → FLAG	
#> Greater than	#03D83h
<bint> <bint> → FLAG	
#< Less than	#03CE4h
<bint> <bint> → FLAG	

The following objects test the value of a single internal binary integer:

#0= Returns TRUE if bint = <0> <div><bint> → FLAG</div>	#03CA6h
#0<> Returns TRUE if bint ≠ <0> <div><bint> → FLAG</div>	#03CC7h
#1= Returns TRUE if bint = <1> <div><bint> → FLAG</div>	#622A7h
#1<> Returns TRUE if bint ≠ <1> <div><bint> → FLAG</div>	#622B6h
#2= Returns TRUE if bint = <2> <div><bint> → FLAG</div>	#6229Ah
#2<> Returns TRUE if bint ≠ <2> <div><bint> → FLAG</div>	#636C8h
#3= Returns TRUE if bint = <3> <div><bint> → FLAG</div>	#62289h
#5= Returns TRUE if bint = <5> <div><bint> → FLAG</div>	#636B4h

Extended Real Number Tests. The following objects compare two extended real numbers:

$%%>$ Greater than $%% \quad %% \rightarrow \text{FLAG}$	#2A87Fh
$%%<$ Less than $%% \quad %% \rightarrow \text{FLAG}$	#2A81Fh
$%%>=$ Greater than or equal to $%% \quad %% \rightarrow \text{FLAG}$	#2A895h
$%%<=$ Less than or equal to $%% \quad %% \rightarrow \text{FLAG}$	#2A8ABh

The following objects test the value of an extended real number:

$%%0=$ Returns TRUE if %% is equal to $%%0$ $%% \rightarrow \text{FLAG}$	#2A75Ah
$%%0<>$ Returns TRUE if %% is not equal to $%%0$ $%% \rightarrow \text{FLAG}$	#2A7BBh
$%%0>$ Returns TRUE if %% is greater than $%%0$ $%% \rightarrow \text{FLAG}$	#2A788h
$%%0<=$ Returns TRUE if %% is less than or equal to $%%0$ $%% \rightarrow \text{FLAG}$	#2A80Bh

Stack Operations

In general, stack operations – even User-RPL stack operations – are fairly efficient. When a program has been streamlined with CMP48 (page 195), the stack operations presented below provide a speed increase.

2DUP5ROLL	#63C40h
ob ₁ ob ₂ ob ₃ → ob ₂ ob ₃ ob ₂ ob ₃ ob ₁	
2DUPSWAP	#611F9h
ob ₁ ob ₂ → ob ₁ ob ₂ ob ₂ ob ₁	
2OVER	#63FBAh
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₁ ob ₂ ob ₃ ob ₄ ob ₁ ob ₂	
2SWAP	#62001h
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₃ ob ₄ ob ₁ ob ₂	
3PICK3PICK	#63C68h
ob ₁ ob ₂ ob ₃ → ob ₁ ob ₂ ob ₃ ob ₁ ob ₂	
3PICKOVER	#630B5h
ob ₁ ob ₂ ob ₃ → ob ₁ ob ₂ ob ₃ ob ₁ ob ₃	
3PICKSWAP	#62EDFh
ob ₁ ob ₂ ob ₃ → ob ₁ ob ₂ ob ₁ ob ₃	
4PICKOVER	#630C9h
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₁ ob ₂ ob ₃ ob ₄ ob ₁ ob ₄	
4PICKSWAP	#62EF3h
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₁ ob ₂ ob ₃ ob ₁ ob ₄	
4ROLLDROP	#62864h
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₂ ob ₃ ob ₄	
4ROLLSWAP	#62ECBh
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₂ ob ₃ ob ₁ ob ₄	
4UNROLL3DROP	#6113Ch
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₄	
4UNROLLDUP	#62D09h
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₄ ob ₁ ob ₂ ob ₃ ob ₃	

4UNROLLROT	#63015h
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₄ ob ₂ ob ₃ ob ₁	
5ROLLDROP	#62880h
ob ₁ ob ₂ ob ₃ ob ₄ ob ₅ → ob ₂ ob ₃ ob ₄ ob ₅	
DROPDUP	#627A7h
ob ₁ ob ₂ → ob ₁ ob ₁	
DROPOVER	#63029h
ob ₁ ob ₂ ob ₃ → ob ₁ ob ₂ ob ₁	
DROPROT	#62FC5h
ob ₁ ob ₂ ob ₃ ob ₄ → ob ₂ ob ₃ ob ₁	
DROPSWAP	#6270Ch
ob ₁ ob ₂ ob ₃ → ob ₂ ob ₁	
DUP4UNROLL	#61099h
ob ₁ ob ₂ ob ₃ → ob ₃ ob ₁ ob ₂ ob ₃	
DUPROT	#62FB1h
ob ₁ ob ₂ → ob ₂ ob ₂ ob ₁	
NDROP	#0326Eh
ob _n ... ob ₁ <n> →	
NDUP (Duplicates <n> objects as a group)	#031D9h
ob _n ... ob ₁ <n> → ob _n ... ob ₁ ob _n ... ob ₁	
NDUPN (Makes <n-1> copies of an object)	#5E370h
ob <n> → ob ... ob <n>	
OVER5PICK	#63C90h
ob ₁ ob ₂ ob ₃ ob ₄ ob ₅ → ob ₁ ob ₂ ob ₃ ob ₄ ob ₅ ob ₄ ob ₂	
OVERDUP	#62CCDh
ob ₁ ob ₂ → ob ₁ ob ₂ ob ₁ ob ₁	
OVERSWAP	#62D31h
ob ₁ ob ₂ → ob ₁ ob ₁ ob ₂	

PICK	$ob_n \dots ob_1 <n> \rightarrow ob_n \dots ob_1 ob_n$	#032E2h
ROLL	$ob_n \dots ob_1 <n> \rightarrow ob_{n-1} \dots ob_1 ob_n$	#03325h
ROLLDROP	$ob_n \dots ob_1 <n> \rightarrow ob_{n-1} \dots ob_1$	#62F89h
ROT2DROP	$ob_1 ob_2 ob_3 \rightarrow ob_2$	#62726h
ROT2DUP	$ob_1 ob_2 ob_3 \rightarrow ob_2 ob_3 ob_1 ob_3 ob_1$	#62C7Dh
ROTDROP	$ob_1 ob_2 ob_3 \rightarrow ob_2 ob_3$	#60F21h
ROTDROPSWAP	$ob_1 ob_2 ob_3 \rightarrow ob_3 ob_2$	#60F0Eh
ROTDUP	$ob_1 ob_2 ob_3 \rightarrow ob_2 ob_3 ob_1 ob_1$	#62775h
ROTOVER	$ob_1 ob_2 ob_3 \rightarrow ob_2 ob_3 ob_1 ob_3$	#62CA5h
ROTROT2DROP	$ob_1 ob_2 ob_3 \rightarrow ob_3$	#6112Ah
ROTSWAP	$ob_1 ob_2 ob_3 \rightarrow ob_2 ob_1 ob_3$	#60EE7h
SWAP2DUP	$ob_1 ob_2 \rightarrow ob_2 ob_1 ob_2 ob_1$	#6386Ch
SWAP3PICK	$ob_1 ob_2 ob_3 \rightarrow ob_1 ob_3 ob_2 ob_1$	#63C54h
SWAP4PICK	$ob_1 ob_2 ob_3 ob_4 \rightarrow ob_1 ob_2 ob_4 ob_3 ob_1$	#63C7Ch
SWAP4ROLL	$ob_1 ob_2 ob_3 ob_4 \rightarrow ob_2 ob_4 ob_3 ob_1$	#63C2Ch

SWAPDROP	$ob_1\ ob_2 \rightarrow ob_2$	#60F9Bh
SWAPDROPDUP	$ob_1\ ob_2 \rightarrow ob_2\ ob_2$	#62830h
SWAPDUP	$ob_1\ ob_2 \rightarrow ob_2\ ob_1\ ob_1$	#62747h
SWAPOVER	$ob_1\ ob_2 \rightarrow ob_2\ ob_1\ ob_2$	#61380h
SWAPROT	$ob_1\ ob_2\ ob_3 \rightarrow ob_3\ ob_2\ ob_1$	#60F33h
UNROLL	$ob_n \dots ob_1\ <n> \rightarrow ob_1\ ob_n \dots ob_2$	#0339Eh
UNROT	$ob_1\ ob_2\ ob_3 \rightarrow ob_3\ ob_1\ ob_2$	#60FACh
UNROTDROP	$ob_1\ ob_2\ ob_3 \rightarrow ob_3\ ob_1$	#6284Bh
UNROTDUP	$ob_1\ ob_2\ ob_3 \rightarrow ob_3\ ob_1\ ob_2\ ob_2$	#62CF5h
UNROTOVER	$ob_1\ ob_2\ ob_3 \rightarrow ob_3\ ob_1\ ob_2\ ob_1$	#6308Dh
reversym	$ob_n \dots ob_1\ <n> \rightarrow ob_1 \dots ob_n\ <n>$	#5DE7Dh

Graphics Operations

The HP 48 contains three built-in graphics objects (nicknamed *grobs*) for use in the display. These grobs are the *stack grob*, the *graphics grob* (*PICT*), and the *menu grob*.

Pointers to Display Grobs. The following objects return pointers to these built-in grobs. If the returned grob is to be modified for later use, you should use NEWOB to create a unique copy in temporary memory.

ABUFF	#12655h
Returns a pointer to the stack display → grob	
GBUFF	#12665h
Returns a pointer to the graphics display → grob	
HARDBUFF	#12635h
Returns a pointer to the currently displayed ABUFF or GBUFF grob → grob	
HARDBUFF2	#12645h
Returns a pointer to the menu grob. → grob	

These pointers are useful for using the stack or graphics display as arguments to other commands. For instance, to place a copy of the stack display in *PICT*, execute

« #12655h SYSEVAL PICT STO »

which is the same as « LCD→ PICT STO » except that the menu is not included.

Display Control. The following objects control the display:

TOADISP Switches the LCD to the stack display →	#1314Dh
TOGDISP Switches the LCD to the graphics display →	#13135h
ClrDA1IsStat Suspends the ticking clock display →	#39531h
RECLAIMDISP Switches LCD to stack display, clears the stack display, and resizes the stack display to 131x56 pixels →	#130ACh
TURNMENUON Turns on the menu display →	#4E347h
TURNMENUOFF Turns off the menu display and enlarges the stack display to 131x64 →	#4E2CFh

The program TOTEXT establishes complete control over a clear stack display:

TOTXT 66 Bytes Checksum #4265h

⌘

#39531h SYSEVAL

Executes ClrDA1IsStat

#130ACh SYSEVAL

Executes RECLAIMDISP

#4E2CFh SYSEVAL

Executes TURNMENUOFF

⌘

Clearing the Display. The following objects clear either part or all of the currently displayed grob (ABUFF or GBUFF). Display rows (pixel rows) are numbered beginning with row number 0 at the top down to 55 at the bottom.

BLANKIT	#126DFh
Clears a specified number of rows <startrow> <# of rows> →	
BlankDA1	#3A546h
Clears rows 0 – 15 →	
BlankDA12	#3A578h
Clears rows 0 – 55 →	
BlankDA2	#3A55Fh
Clears rows 16 – 55 →	
CLEARVDISP	#134AEh
Zeros out the entire grob pointed to by HARDBUFF →	
Clr8	#0E083h
Clears the top 8 rows →	

Clearing Part of a Grob. The object GROB!ZERO clears a rectangle from x_1, y_1 in the upper-left to x_2, y_2 , which is one pixel below and to the right of the area to be cleared. The upper-left corner of a grob has the coordinates $\langle 0 \rangle \langle 0 \rangle$.

GROB!ZERO

#11A6Dh

Zeros out a rectangular portion of a grob.

grob $\langle x_1 \rangle \langle y_1 \rangle \langle x_2 \rangle \langle y_2 \rangle \rightarrow$ grob

WARNING

GROB!ZERO does no range checking, and will write beyond the boundaries of the grob if given incorrect coordinates. In that event, memory will be corrupted.

The grob parameter may be any arbitrary grob, including HARDBUFF. GROB!ZERO writes directly to memory, and the pointer to the grob is returned to level 1.

Note that *PICT* is not a valid parameter for the target grob for the command GROB!ZERO. To erase a portion of the stack display or graphics display, use ABUFF or GBUFF.

Creating a Blank Grob. The User-RPL command BLANK takes two user binary integers (object type 10) from the stack and creates a blank graphics object of the specified size. The System-RPL object MAKEGROB does the work for BLANK. The input parameters are internal binary integers (object type 20):

MAKEGROB	#1158Fh
Creates a blank grob	
<height> <width> → grob	

Finding the Size of a Grob. The objects GROBDIM and DUPGROBDIM return the size of a grob expressed in internal binary integers:

GROBDIM	#50578h
Returns the dimensions of a grob	
grob → <height> <width>	
DUPGROBDIM	#5179Eh
Duplicates a grob before returning its dimensions	
grob → grob <height> <width>	

Adding Graphics Objects to the Stack Display. The object XYGROBDISP is used to place a grob into the grob pointed to by HARDBUFF (the stack display or *PICT* – see page 179).

XYGROBDISP	#128B0h
Adds a grob to HARDBUFF	
<code><x> <y> grob →</code>	

Coordinates are specified with internal binary integers, and use pixel coordinates (see *Graphics Coordinates*). The upper-left corner of the display has the coordinates $x=<0> y=<0>$. If the grob being added to HARDBUFF would extend beyond the boundaries of HARDBUFF, then HARDBUFF is enlarged to accomodate the new grob.

The program XYGD takes a grob in level 3, x -coordinate as a real number in level 2, and y -coordinate as a real number in level 1, and displays the grob in HARDBUFF:

XYGD 45 Bytes Checksum #5095h

grob x y →

⌘

CRCE2	<i>Convert real coordinates into bints</i>
ROT	<i>Order parameters for XYGROBDISP</i>
#128B0h SYSEVAL	<i>Execute XYGROBDISP</i>

⌘

To glue several of these examples together, try the program FUN:

FUN 95.5 Bytes Checksum #C47h

⌘

TOTXT	<i>Take over the stack display</i>
"SYSTEM PROGRAMMING IS FUN!"	
1 →GROB	<i>Convert text to a grob</i>
11 30 XYGD	<i>Place grob in stack display</i>
7 FREEZE	<i>Freeze the display</i>

⌘

Making a Box. The object GROB!ZERO may be used to create a blank grob with a one-pixel-wide border around it. The program MKBOX takes the desired height (in pixels expressed in real numbers) of the box from level 2 and the width from level 1.

MKBOX 131 Bytes Checksum #7805h

height width → grob

«

CRCE2	<i>Convert the real numbers into bints</i>
#1158Fh SYSEVAL NEG	<i>Create the grob and invert it</i>
#63AC4h SYSEVAL	<i>Put the bints <1> <1> on the stack</i>
3 PICK	<i>(4: grob 3: <1> 2: <1> 1: grob)</i>

Now get the grob dimensions for the lower-right corner with the GROBDIM command

#50578h SYSEVAL	<i>(5: grob 4: <1> 3: <1> 2: <y> 1: <x>)</i>
#3E0Eh SYSEVAL	<i>Subtract <1> from width</i>
SWAP	<i>Swap to create x-coordinate</i>
#3E0Eh SYSEVAL	<i>Subtract <1> from height</i>
#11A6Dh SYSEVAL	<i>Zero out the interior of the grob</i>

»

Now use the program SHWBX to create a box of arbitrary size and show it in the upper-left corner of the stack display:

SHWBX 54 Bytes Checksum #4C38

height width →

«

TOTXT	<i>Take over the stack display</i>
MKBOX	<i>Create the box</i>
0 0 XYGD	<i>Show the box in the upper left</i>
7 FREEZE	<i>Freeze the display</i>

»

Placing Text in a Box. Suppose you’ve drawn a graph in *PICT*, and you wish to place a nice title at the top of the graph. The programs TXTBX and LBLPICT may be used to get the job done. The first program, TXTBX, accepts a string in level 2 and a font number (1=small, 2=medium, 3=large) in level 1.

TXTBX 89 Bytes Checksum #545Fh

“string” font_number → grob

⌘	→GROB	<i>Convert text to grob</i>
	DUP SIZE	<i>Get size as user binary integers</i>
	B→R 3 +	<i>Calculate height of box as a real</i>
	SWAP B→R 3 +	<i>Calculate width of box as a real</i>
	MKBOX	<i>Create the box with a border</i>
	{ #2h #2h }	<i>Coordinates for text insertion</i>
	ROT GOR	<i>Place text in the box</i>
⌘		

Combining Graphics Objects. The User-RPL commands GOR and GXOR are useful for superimposing one grob on top of another, but when you want to place a smaller grob into a larger grob as a replacement operation, the User-RPL command REPL or the System-RPL object GROB! become very useful.

GROB! #11679h
 Stores level 4 grob into level 3 grob at specified coordinates
 grob grob <col> <row> →

WARNING

GROB! does no range checking, and will write beyond the boundaries of the target grob if given incorrect coordinates. In that event, memory will be corrupted.

Note that *PICT* is not a valid parameter for the target (level 3 input parameter) grob for the command GROB! To write to the stack display or graphics display, use ABUFF or GBUFF.

Now you can use the program LBLPICT to place a boxed label at the top center of *PICT*. LBLPICT takes the same parameters as TXTBX:

LBLPICT 150 Bytes Checksum #FA1h

"string" font_number →

⌘

TXTBX

#12665h SYSEVAL

DUP SIZE DROP

3 PICK SIZE DROP

IF DUP2 2 + <

THEN

"Label Too Big" DOERR

END

- 2 /

B→R 0 CRCE2

#11679h SYSEVAL

{ } PVIEW

⌘

Create the label in a box

Get pointer to graphics display

Find the width of graphics display

Get the real width of the label box

Make sure the label will fit

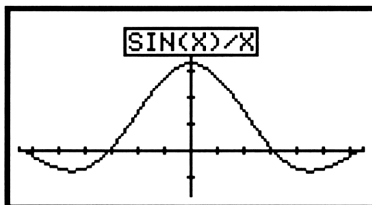
Calculate x position for label

Assemble coordinates for GROB!

Add the label

*Display PICT, wait for **CANCEL***

To try out LBLPICT, plot the function $\sin(X)/X$ in radians mode with the range set to $(-6.5 \leq X \leq 6.5)$, $(-.5 \leq Y \leq 1.4)$, then execute LBLPICT with the parameters "SIN(X)/X" 2:



The program ⌘ EQ →STR 2 LBLPICT ⌘ will label the graph in *PICT* with the contents of EQ (assuming that the equation is fairly short).

Drawing Lines. Several System-RPL objects are available for line drawing. Lines are drawn from coordinates $\langle x_1 \rangle, \langle y_1 \rangle$ to $\langle x_2 \rangle, \langle y_2 \rangle$. The coordinates for these line drawing objects are expressed as bints and *require* that input coordinates be placed on the stack in left-to-right order – that is, $\langle x_1 \rangle \leq \langle x_2 \rangle$. The command ORDERXY# is useful for ordering the input parameters to the line drawing objects:

ORDERXY#	#51893h
Asserts left-to-right order for line-drawing coordinates	
$\langle x_1 \rangle \ \langle y_1 \rangle \ \langle x_2 \rangle \ \langle y_2 \rangle \rightarrow \langle x_1 \rangle \ \langle y_1 \rangle \ \langle x_2 \rangle \ \langle y_2 \rangle$	

The program ORDXY accepts real-number coordinates, converts them to bints, and executes ORDERXY# to prepare for the internal line drawing commands. The stack diagram for ORDXY is:

ORDXY 67.5 Bytes Checksum #B2EDh

$x_1 \ y_1 \ x_2 \ y_2 \rightarrow \langle x_1 \rangle \ \langle y_1 \rangle \ \langle x_2 \rangle \ \langle y_2 \rangle$

⌘

CRCE2

Convert coordinates to bints

#62001h SYSEVAL

Swap coordinate pairs with 2SWAP

CRCE2

Convert other coordinate pair

#51893h SYSEVAL

Evaluate ORDERXY#

⌘

The User-RPL commands LINE and TLINE draw a solid line or a line which toggles pixel states in the graphics display. The System-RPL object LINEOFF3 clears a line of pixels:

LINEOFF3	#50ACCh
Clears a line of pixels in the graphics display	
$\langle x_1 \rangle \ \langle y_1 \rangle \ \langle x_2 \rangle \ \langle y_2 \rangle \rightarrow$	

The System-RPL objects LINEON, LINEOFF, and TOGGLE draw lines in the stack display:

LINEON	#50B17h
Draws a line in the stack display $\langle x_1 \rangle \langle y_1 \rangle \langle x_2 \rangle \langle y_2 \rangle \rightarrow$	
LINEOFF	#50B08h
Clears a line of pixels in the stack display $\langle x_1 \rangle \langle y_1 \rangle \langle x_2 \rangle \langle y_2 \rangle \rightarrow$	
TOGGLE	#50AF9h
Toggles a line of pixels in the stack display $\langle x_1 \rangle \langle y_1 \rangle \langle x_2 \rangle \langle y_2 \rangle \rightarrow$	

Pixel Control. The User-RPL commands PIXON, PIXOFF, and PIX? clear, set, and test pixels in the graphics display. Their System-RPL counterparts for the stack display have similar names:

PIXON	#1384Ah
Turns on a pixel in the stack display $\langle x \rangle \langle y \rangle \rightarrow$	
PIXOFF	#1383Bh
Turns off a pixel in the stack display $\langle x \rangle \langle y \rangle \rightarrow$	
PIXON?	#13992h
Tests a pixel in the stack display $\langle x \rangle \langle y \rangle \rightarrow \text{FLAG}$	

The PIXON? object returns an internal flag, which may be converted into a real number 0 or 1 with the System-RPL command COERCEFLAG.

Menu Graphics. The following objects create menu label grobs (8 pixels high by 21 pixels wide) given a string as input:

MakeStdLabel Creates a standard label “string” → grob	#3A328h
MakeDirLabel Creates a directory label “string” → grob	#3A3ECh
MakeBoxLabel Creates a label with a “mode box” at the right side “string” → grob	#3A38Ah
MakeInvLabel Creates an outline box label “string” → grob	#3A44Eh

The objects DispMenu and DispMenu.1 display the current menu:

DispMenu Displays the current menu and freezes the menu display line →	#3A1E8h
DispMenu.1 Displays the current menu →	#3A1FCh

These objects are useful for updating the menu display without using the HALT or WAIT commands:

```
« ... { menu } TMENU #3A1FCh SYSEVAL ... »
```

Keyboard Control

Several objects are available to provide extra control over key detection and maintenance of the key buffer.

Waiting for a Key. The User-RPL command WAIT returns a key location in RC.P format, but does not return the **CANCEL** key. If you wish to trap the **CANCEL** key, WaitForKey is the right tool for the job. WaitForKey places the HP 48 in light sleep to conserve batteries, processes alarms, and waits for keyboard activity. WaitForKey processes the shift keys, maintains the shift annunciators, and returns the key press expressed as two bints – a *shift plane* and a *keycode*.

WaitForKey	#41F65h
Waits for a fully formed keyboard event	
→ <keycode> <plane>	

Keycodes are returned in the range <1d> to <49d> (starting in the upper-left corner of the keyboard), and planes are returned in the range 1 to 6:

Plane	Description
<1d>	Unshifted
<2d>	Left-shifted
<3d>	Right-shifted
<4d>	Alpha
<5d>	Alpha left-shifted
<6d>	Alpha right-shifted

To convert a result to RC.P format (suitable for user key assignments), use the object CodePl>%rc.p:

CodePl>%rc.p	#41D92h
Converts keycode and plane bints into real number rc.p key address	
<keycode> <plane> → %rc.p	

The Key Buffer. The object KEYINBUFFER returns an internal flag that indicates whether a key has been pressed.

KEYINBUFFER?	#42402h
Returns TRUE if a key is waiting in the keybuffer	
→ FLAG	

Use the object COERCEFLAG to convert the internal flag into a real number 0 or 1. Remember that the call to COERCEFLAG needs to be enclosed in an extra set of program delimiters.

The program CKKB uses the objects KEYINBUFFER? and COERCEFLAG to check the state of the key buffer. CKKB returns the real number 0 if no key has been pressed, or 1 if a key has been pressed:

CKKB 64.5 Bytes Checksum #43E7h

→ T/F

⌘

#42402h SYSEVAL

Check the key buffer

⌘ #5380Eh SYSEVAL ⌘ EVAL

Convert flag to 0 or 1

⌘

To detect whether the **CANCEL** key has been pressed, use the object ATTN?. This object will not affect the key buffer.

ATTN?	#42262h
Returns TRUE if CANCEL has been pressed.	
→ FLAG	

Alpha-Lock. The INPUT command provides an option to specify alpha-lock on the keyboard, but there may be other times that you wish to insure that alpha entry mode is either enabled or disabled. Two objects are available for this:

LockAlpha	#40D25h
Locks alpha entry mode	
→	
UnLockAlpha	#40D39h
Unlocks alpha entry mode	
→	
1A/LockA	#3AA0Ah
Locks alpha entry for one keystroke (same as pressing <input type="checkbox"/> once)	
→	

The MatrixWriter. The MatrixWriter can be started from a program by executing either DoNewMatrix to create a new matrix or DoOldMatrix to edit a matrix on the stack.

DoNewMatrix	#44C31h
Starts the MatrixWriter	
→ [[matrix]]	<i>If terminated with</i> <input type="button" value="ENTER"/>
→	<i>If terminated with</i> <input type="button" value="CANCEL"/>
DoOldMatrix	#44FE7h
Edits matrix	
[[matrix]] → [[matrix]]' TRUE	<i>If terminated with</i> <input type="button" value="ENTER"/>
[[matrix]] → FALSE	<i>If terminated with</i> <input type="button" value="CANCEL"/>

Streamlining Finished Programs

Once you have perfected a new program that you plan to use frequently, you may want to make the program smaller and faster. The use of system objects can be streamlined by replacing the user binary integers and SYSEVAL commands with a single 2-1/2 byte address of the proper System-RPL object. This can be done with the program CMP48. Once streamlined, the program will be smaller and faster.

The core of the streamlining strategy is embodied in a small piece of assembly language code which is created by converting a string object into an assembly language code object. This operation is critical, so *please* double check the checksum of CMP48 before you use it.

The code object and the SYSEVAL command are stored in local variables *syseval* and *code*, then each object in the program is examined to see if a conversion is needed. The procedure used is similar to the program example SFILT in the section *Meta-Objects*. If a SYSEVAL command is found, it is dropped and its address argument is converted into an object address. If a program object is found, it is streamlined using CMP48. Note that CMP48 fails if an address is not found before the SYSEVAL command.

This recursive approach carries a minor inefficiency – each time CMP48 is executed, the code object is recreated. This program accepts this cost to permit the entire process to be contained in a brief program that can be entered from the keyboard.

To use CMP48, place the program to be streamlined into level 1 and execute CMP48. The resulting program may be stored in a variable and executed like any other program. When the streamlined program is displayed on the stack, the SYSEVAL commands are replaced by the word External.

WARNING

Once a program has been streamlined, *it cannot be edited*. For this reason, it's a good idea to keep a copy of the original program in a safe place. When in doubt, back up your HP 48 before using CMP48.

«	
IF DUP TYPE 8 ≠	<i>Is this a program?</i>
THEN 514 DOERR	<i>Bad Argument Type if not</i>
END	
0 SWAP	<i>Empty meta-ob in pos 2</i>
NEWOB	<i>Ensure program is unique</i>
#54AFh SYSEVAL	INNERCOMP
#18DBFh SYSEVAL	UNCOERCE
{ SYSEVAL } 1 GET	<i>Place SYSEVAL on stk</i>
200 8 70 18 65 20 21	<i>Code object data</i>
49 52 20 113 115 23 65	
" 1 14 START	<i>Accumulate data into</i>
SWAP CHR + NEXT	<i>a string</i>
11724 #18CEAh SYSEVAL	COERCE code prologue
#5AB3h SYSEVAL	CHANGETYPE
→ syseval code	
«	
WHILE DUP 0 ≠	<i>Loop for every ob in prog</i>
REPEAT	
1 - SWAP	MDT
IF DUP TYPE 8 ==	<i>If this is a program</i>
THEN CMP48	<i>then compact it too</i>
ELSE	
IF DUP syseval ==	<i>If we have a SYSEVAL</i>
THEN DROP 1 - SWAP	<i>throw it away, get address</i>
B→R #18CEAh SYSEVAL	<i>and COERCE it to a bint</i>
code EVAL	<i>Convert bint to address</i>
END	
END	
OVER DUP 4 + PICK + 3 +	MAH2
ROLLD DUP 2 + ROLL 1 +	
OVER 2 + ROLLD	
END DROP	<i>Loop end</i>
#18CEAh SYSEVAL	COERCE
#5445h SYSEVAL	::N
»	
»	

The data for the string has been compiled using HP's SASM assembler (see *HP Tools*). The object CHANGETYPE (#05AB3h) requires an object in level two and a bint with the new object prologue in level 1.

To satisfy the advanced user of the HP tools and the curious, the assembly language source code for the original code object reads:

```
NIBASC      \HHP48-A\  
NIBHEX      CCD20  
REL (5)     end  
C=DAT1      A  
CD1EX  
D1=D1+      5  
A=DAT1      A  
D1=C  
DAT1=A      A  
A=DAT0      A  
D0=D0+      5  
PC= (A)
```

end

When assembled with the SASM assembler mentioned below, this code produces the code object used by CMP48. The code should be assembled with the command `SASM -H filename`.

LIBEVAL

Approximately half of the HP 48 operating system is implemented with *libraries*, which requires a different access method. Access to objects in libraries is possible through *XLIB names* (object type 14). To create an XLIB name, use the system object #>ROMPTR:

#>ROMPTR	#7E50h
Creates an XLIB name from two internal binary integers	
<library no.> <object no.> → XLIB_name	

You can evaluate a library object by placing two internal binary integers on the stack, executing `#>ROMPTR`, then `EVALing` the resulting XLIB name.

The function `LIBEVAL` simplifies the process used to evaluate objects in a library. The parameter to `LIBEVAL` is a user binary integer, just like `SYSEVAL`. In the case of `LIBEVAL`, the integer must contain six (hex) digits. The upper three digits specify the library number, and the lower three digits specify the number of the object within the library. Objects in libraries are numbered beginning with 0. It is important to remember that `LIBEVAL` is a command like any other, and saves its last argument (if flag `-55` is clear), *not* the arguments processed by the system object.

Example: The Multiple Equation Solver saves its equations, title, variable status, and progress information in the reserved variable *Mpar*. This variable is a Library Data object, and its contents are not visible. To recall the contents of *Mpar*, execute the `MESRclEqn`, which is the 18th object in library E4 (XLIB 228 18):

MESRclEqn	#E4012h	XLIB 228 18
Recalls the contents of the reserved variable <i>Mpar</i> → { equation list }		

`#0E4012h LIBEVAL → { equation list }`

Example: The program `MSGBOX` (on the next page) uses the system object `DoMsgBox` (XLIB 177 0) to display a message box with an included graphics object. `MSGBOX` requires a string in level 2 and a grob in level 1. In theory, the grob may be as wide as the message box, but in practice a smaller grob will leave more room for message text.

DoMsgBox	#B1000h	XLIB 177 0
Displays a message box with a graphics object “message” #maxwidth #minwidth grob menuob → TRUE		

The parameters are defined as follows:

- “message” A string containing the message you wish to display. Carriage-returns may be embedded to force line breaks.
- #maxwidth An internal binary integer specifying the maximum character width of each text line in the message box.
- #minwidth An internal binary integer specifying the minimum character width of each text line in the message box. No word breaks occur before the minimum character width.
- grob A graphics object to be displayed in the upper-left corner of the message box.
- menuob A message box menu object, usually specified in the form XLIB 177 2. Library object names (object type 14) can be created using the system object #>ROMPTR.

DoMsgBox returns the internal flag TRUE. You may wish to experiment with different values for the minimum and maximum character widths. Neither value should exceed 15. Remember to leave room for the grob.

MSGBOX 123.5 Bytes Checksum #E966h

“string” grob →

⌘

12	<i>Maximum character width</i>
10	<i>Minimum character width</i>
#194F7h SYSEVAL	<i>Use COERCE2 to convert to bints</i>
ROT	<i>Move grob into position</i>
177 2	<i>Menu object is in library 177, function 2</i>
#194F7h SYSEVAL	<i>Use COERCE2 to convert to bints</i>
#7E50h SYSEVAL	<i>Use #>ROMPTR to convert 2 bints to XLIB</i>
#B1000h LIBEVAL	<i>Execute DoMsgBox</i>
DROP	<i>Drop returned TRUE flag</i>

⌘

Try MSBGX with the program TRYMBX:

TRYMBX 87 Bytes Checksum #F68Fh

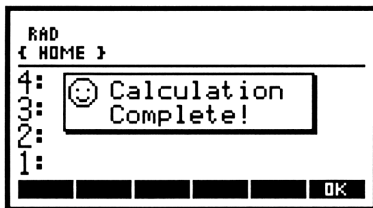
«

"Calculation Complete!"

GROB 11 11 8F004010202010409840104010409840272040108F00

MSBGX

»



HP Tools

Hewlett-Packard has developed tools for HP 48 software development that run on a DOS-compatible personal computer. These tools provide documentation and programming possibilities beyond those discussed in this chapter. The tools are:

RPLCOMP	A compiler for the internal RPL language
SASM	The Saturn assembler
SLOAD	The Saturn loader
MAKEROM	A utility for building libraries from System-RPL code
USRLIB	A utility for building libraries from user directories

These tools and documentation are provided on an “as-is” basis, and are not supported by HP (*or* the author). You can find these tools on the bulletin board mentioned in the HP 48 User’s Guide.

System Operations

Invoking System Operations

To invoke a system operation, press and hold **[ON]**, then press and release the second key, then release **[ON]**.

[ON] [A] and [F] Erases all memory (including port 0 and merged memory) and performs a warmstart (see **[ON] [C]**). Merged memory remains merged.

[ON] [B] Cancels the current selection if selected before all keys are released.

[ON] [C] Often known as a *system halt* or a *warmstart*, this operation places the calculator in a known state without resetting user memory. The stack is cleared, the VAR directory is set to HOME, the MTH menu is displayed, User mode is cleared, *PICT* is cleared, and the system configuration is updated to recognize all libraries.

[ON] [D] Starts the interactive self test (see below).

[ON] [E] Runs a continuous self test.

[ON] [SPC] Coma mode: a deep-sleep shutdown which turns off the system timers (including the clock) and clears the system halt log.

[ON] [I/O] Performs a graphics screen dump in HP 82240A/B graphics format (regardless of the I/O port selection).

[ON] [+ or -] Adjusts the display contrast

[ON] [TIME] Cancels the next repeating alarm.

System Halt Log

The command WSLOG returns four strings to the stack showing the cause, date, and time of the four most recent system halt events. The system halt log is not cleared when memory is erased, and may only be cleared by placing the calculator in coma mode.

Example: 3-03/06/90 09:30:10

This string shows a type 3 system halt that occurred on the morning of March 6, 1990.

Code	Condition
0	Coma exit
1	Low battery system save
2	I/O timeout
3	Execute through address 0
4	Corrupt system time
5	Port change data
7	Corrupt CMOS test word
8	Hardware failure
9	Corrupt alarm list
A	Corrupt memory
B	Module pulled
C	Hardware reset
D	Missing RPL error handler
E	Corrupt configuration table
F	System RAM card pulled

Note: Some events will cause two events to be recorded, and some system halt events will cause memory to be cleared (see **ON** **A** **F**).

Interactive Self Test

The **[ON][D]** sequence enters the HP 48 interactive self test. Once the test has been started, there are a variety of options:

[A]	Displays the CPU speed
[B]	Press [ENTER] for display test patterns
[C]	Internal ROM check
[D]	Internal RAM check
[E]	Keyboard test
[F]	Partial keyboard test
[G]	ESD test monitor. Bars indicate battery status.
[H]	UART loop back test
[I]	Wired UART echo
[J]	Shows what's plugged in
[K]	Test port RAM devices
[L]	Blank display
[M]	Send system time from IR port
[N]	Receive system time from IR port
[O]	Wireless loop back
[P]	Wireless UART echo
[S]	Show test start time
[T]	Show test fail time
[U]	Looping test
[V]	Looping test
[W]	Looping test
[X]	Looping test
[ENTER]	Initialize test times
[Y]	Looping test
[Z]	Looping test
[DEL]	Test summary

Press **[ON][C]** (system halt) to return to the stack display.

Statistics Data

Data used by the **STAT** application resides in or is named by the reserved variable ΣDAT . Statistics data may be entered from the stack one point at a time using $\boxed{\text{STAT}} \boxed{\text{DATA}} \boxed{\Sigma+}$, or an entire matrix can be stored into ΣDAT using the $\text{STO}\Sigma$ command. The command $\text{EDIT}\Sigma$ may be used to edit ΣDAT using the MatrixWriter.

$\boxed{\Sigma+}$ and $\boxed{\Sigma-}$ operate as follows:

X	$\boxed{\Sigma+}$ → Append one data point with one coordinate value
	$\boxed{\Sigma-}$ → Reverses the effect of the last $\Sigma+$
$[X_1 \ X_2 \ \dots \ X_m]$	$\boxed{\Sigma+}$ → Append one data point with m coordinate values
$[[X_{11} \ X_{12} \ \dots \ X_{1m}]$	
\dots	
$[X_{n1} \ X_{n2} \ \dots \ X_{nm}]]$	$\boxed{\Sigma+}$ → Append n data points with m coordinate values

ΣDAT

The variable ΣDAT contains either a statistics data matrix or the name of a variable containing a statistics data matrix.

ΣDAT Statistics Matrix						
Data Point	Coordinate Number					
	1	2	3	4	...	m
1	X_{11}	X_{12}	X_{13}	X_{14}	...	X_{1m}
2	X_{21}	X_{22}	X_{23}	X_{24}	...	X_{2m}
3	X_{31}	X_{32}	X_{33}	X_{34}	...	X_{3m}
...
n	X_{n1}	X_{n2}	X_{n3}	X_{n4}	...	X_{nm}

Σ PAR

The reserved variable ΣPAR contains plot and scaling information. Each directory may contain a unique ΣPAR . The entries for the independent and dependent columns may be set using the COL Σ command.

Σ PAR		
{ indep dep intercept slope model }		
Parameter	Description	Default
indep	Independent column number	1
dep	Dependent column number	2
intercept	Intercept of current regression model	0
slope	Slope of current regression model	0
model	Current model: LINFIT, EXPFIT, PWRFIT, or LOGFIT	LINFIT

Character Codes

DEC	HEX	CHR	DEC	HEX	CHR	DEC	HEX	CHR	DEC	HEX	CHR
0	00	▪	32	20		64	40		96	60	'
1	01	▪	33	21	!	65	41		97	61	a
2	02	▪	34	22	"	66	42	B	98	62	b
3	03	▪	35	23	#	67	43	C	99	63	c
4	04	▪	36	24	\$	68	44	D	100	64	d
5	05	▪	37	25	%	69	45	E	101	65	e
6	06	▪	38	26	&	70	46	F	102	66	f
7	07	▪	39	27	'	71	47	G	103	67	g
8	08	▪	40	28	(72	48	H	104	68	h
9	09	▪	41	29)	73	49	I	105	69	i
10	0A	▪	42	2A	*	74	4A	J	106	6A	j
11	0B	▪	43	2B	+	75	4B	K	107	6B	k
12	0C	▪	44	2C	,	76	4C	L	108	6C	l
13	0D	▪	45	2D	-	77	4D	M	109	6D	m
14	0E	▪	46	2E	.	78	4E	N	110	6E	n
15	0F	▪	47	2F	/	79	4F	O	111	6F	o
16	10	▪	48	30	0	80	50	P	112	70	p
17	11	▪	49	31	1	81	51	Q	113	71	q
18	12	▪	50	32	2	82	52	R	114	72	r
19	13	▪	51	33	3	83	53	S	115	73	s
20	14	▪	52	34	4	84	54	T	116	74	t
21	15	▪	53	35	5	85	55	U	117	75	u
22	16	▪	54	36	6	86	56	V	118	76	v
23	17	▪	55	37	7	87	57	W	119	77	w
24	18	▪	56	38	8	88	58	X	120	78	x
25	19	▪	57	39	9	89	59	Y	121	79	y
26	1A	▪	58	3A	:	90	5A	Z	122	7A	z
27	1B	▪	59	3B	;	91	5B	[123	7B	{
28	1C	▪	60	3C	<	92	5C	\	124	7C	
29	1D	▪	61	3D	=	93	5D]	125	7D	}
30	1E	▪	62	3E	>	94	5E	^	126	7E	~
31	1F	...	63	3F	?	95	5F	_	127	7F	

DEC	HEX	CHR	DEC	HEX	CHR	DEC	HEX	CHR	DEC	HEX	CHR
128	80	€	160	A0		192	C0	À	224	E0	à
129	81		161	A1	í	193	C1	Á	225	E1	á
130	82		162	A2	ª	194	C2	Â	226	E2	â
131	83		163	A3	ë	195	C3	Ã	227	E3	ã
132	84		164	A4	ä	196	C4	Ä	228	E4	ä
133	85		165	A5	ý	197	C5	Å	229	E5	å
134	86		166	A6	î	198	C6	Æ	230	E6	æ
135	87		167	A7	ï	199	C7	Ç	231	E7	ç
136	88		168	A8		200	C8	È	232	E8	è
137	89		169	A9		201	C9	É	233	E9	é
138	8A		170	AA		202	CA	Ê	234	EA	ê
139	8B		171	AB		203	CB	Ë	235	EB	ë
140	8C		172	AC		204	CC	Ì	236	EC	ì
141	8D		173	AD		205	CD	Í	237	ED	í
142	8E		174	AE		206	CE	Î	238	EE	î
143	8F		175	AF		207	CF	Ï	239	EF	ï
144	90		176	B0		208	D0	Ð	240	F0	ð
145	91		177	B1		209	D1	Ñ	241	F1	ñ
146	92		178	B2		210	D2	Ò	242	F2	ò
147	93		179	B3		211	D3	Ó	243	F3	ó
148	94		180	B4		212	D4	Ô	244	F4	ô
149	95		181	B5		213	D5	Õ	245	F5	õ
150	96		182	B6	¡	214	D6	Ö	246	F6	ö
151	97		183	B7	¢	215	D7	×	247	F7	÷
152	98		184	B8	£	216	D8	Ø	248	F8	ø
153	99		185	B9	¤	217	D9	Ù	249	F9	ù
154	9A		186	BA	¥	218	DA	Ú	250	FA	ú
155	9B		187	BB	¦	219	DB	Û	251	FB	û
156	9C		188	BC	§	220	DC	Ü	252	FC	ü
157	9D		189	BD	¨	221	DD	Ý	253	FD	ý
158	9E		190	BE	©	222	DE	Þ	254	FE	þ
159	9F		191	BF	ª	223	DF	ß	255	FF	ÿ

Data Transfer

Any named object, such as a variable, backup object, or complete directory, may be transferred to another HP 48 or a computer. A complete backup of user memory may also be transferred to another HP 48 or a computer.

Note: Binary data, such as programs and libraries, may be transferred from a HP 48S/SX to a HP 48G/GX, but programs or libraries designed for the HP 48G/GX may not work on an HP 48S/SX.

Data Transfer Methods

There are three methods of transferring data between the HP 48 and another HP 48 or computer:

- Objects may be transferred between HP 48s using the infrared (IR) link. The IR link is fixed at 2400 baud, no parity, and may be used to transfer data in either ASCII or binary mode.
- Objects may be transferred between a computer and an HP 48 using the serial (wire) link. The wire link may be configured to support a variety of baud rates and parity options. The Kermit protocol provides the most reliable transfer mechanism, but the Xmodem transfers are fastest for binary downloads of large objects.
- Plug-in RAM cards may be configured as independent memory and exchanged between HP 48s. The commands FREE1, MERGE1, and PINIT are used to configure RAM cards. Only library and backup objects can reside in independent memory.

Kermit Protocol

The *Kermit* file transfer protocol ensures correct data transmission between two HP 48 calculators or an HP 48 and a computer. Kermit was developed at the Columbia University Center for Computing Activities. Detailed information about Kermit is available in a book by Frank da Cruz, *KERMIT, A File Transfer Protocol*, 1987, Bedford, MA (Digital Press). For 9600 baud transfers, it's best to disable the updating clock display.

Kermit Configurations. Kermit protocol provides two basic configurations for data transfer:

- | | |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Local/Local | Commands must be entered on both machines to effect a transfer: a SEND command must be issued on the sender, and a RECEIVE (RECV or RECN on the HP 48) command must be issued on the receiver. New commands must be issued for each object transferred. (Some implementations of Kermit permit “wildcard” characters to send a series of files with one command.) |
| Local/Server | <p>One machine is placed in <i>server</i> mode, which acts upon commands received from the sender. The server:</p> <ul style="list-style-type: none">• Transmits an object when it receives a GET command with a file name.• Receives an object when it receives a SEND command.• Exits Kermit when it receives a FINISH command. |

The server may respond to multiple transfer requests without keyboard intervention.

Remote Kermit Operation. The HP 48 can respond to several Kermit commands when in server mode. These commands initiate actions, list variables, or transfer data.

GET: The Kermit command GET *name* instructs the HP 48 server to transmit the contents of the named variable to the computer.

SEND: The Kermit command SEND *name* instructs the HP 48 server to receive the contents of the named computer file and store them in a variable of the same name.

REMOTE DIR: The Kermit command REMOTE DIR (packet GD) causes the HP 48 server to reply with the current directory path, the number of bytes of free memory, and then a separate line for each variable in the current directory. Each line contains the variable name, length in bytes, type, and a decimal checksum. Examples:

Name	Length	Type	Checksum
X	16	Real Number	7537
EQ	40	Algebraic	14632
CLK	6876	Directory	28291
IOPAR	29.5	List	7079

REMOTE HOST: The Kermit command REMOTE HOST (C "host-command" packet) may be used to execute HP 48 commands from the computer. After the command has been executed, the HP 48 replies by returning the stack contents. The stack is formatted in a manner similar to the PRSTC (print stack compact) command. For instance, to add two numbers on the HP 48, type "REMOTE HOST 2 3 +". Assuming a previously empty stack, the HP 48 replies with the string "1: 5". If the stack is empty, the HP 48 replies "Empty Stack".



FINISH: The Kermit command FINISH transmits the GF packet to the HP 48 to turn off server mode on the HP 48. The GL packet, associated with logout commands, has the same effect.

Backing Up the HP 48

The ARCHIVE and RESTORE commands may be used to save and recover the contents of user memory on a computer.

Note: ARCHIVE does not save the user and system flags, or the contents of port 0. The flag settings may be preserved by executing RCLF and storing the flags in a variable. After doing a restore, recall the contents of the variable and execute STOF.

To back up all of user memory to a computer, perform the following steps:

- Connect the HP 48 and the computer.
- Use the   IOPAR menu to set wire transmission mode, the baud rate, parity, and checksum settings.
- *Optional:* Execute RCLF and store the flags in a variable.
- Enter the object :IO: name, where name is the computer file name that will contain the HP 48 archive. For 9600 baud transfers, it's best to disable the updating clock display.
- Issue the Kermit RECEIVE command on the computer.
- Execute ARCHIVE on the HP 48.

Restoring the HP 48

CAUTION

The RESTORE command erases the *entire* contents of user memory.

To restore HP 48 memory from an archive on a computer, perform the following steps:

- Be sure there is enough memory available to hold the incoming file. Since RESTORE will replace all of user memory, you might as well purge all variables.
- Connect the HP 48 and the computer.
- Transfer the file containing the memory image to the HP 48 the same was as for any file.
- Put the file name on the stack and execute RCL. This puts **Backup HOMEDIR** in level 1.
- Execute RESTORE.
- *Optional:* Recall the variable containing the user and system flags and execute STOF.

ASCII File Transfer

An ASCII file generated on a computer provides an alternative method for entering data or a large program in the HP 48. To ensure that the data is interpreted correctly by the receiving HP 48, the following header string should be included which indicates the expected modes:

`%HP: T<translation>A<angle mode>F<fraction-mark>;`

The codes are defined as follows:

Code	Purpose	Settings	Default
T	See <i>Character Translations</i>	0, 1, 2, or 3	1
A	Sets the angle mode	D, R, or G	D
F	Sets the fraction mark	, or .	.

The HP 48 will ignore text between two @ characters or between an @ character and the end of a line in the computer file.

Example: The following text on a computer may be transferred to the HP 48 in ASCII mode to create a program that returns the area and volume of a sphere given its radius. Notice the use of character translations to represent various HP 48 characters:

```
%HP: T(3)A(D)F(.);
\<< \-> r \<< @ Comment information
  4 \pi \->NUM * r 2 ^ * "Area" \->TAG
  4 3 / \pi \->NUM * r 3 ^ * "Volume" \->TAG
\>>
\>>
```

On the HP 48, the program looks like this:

```
« → r
  « 4 π →NUM * r 2 ^ * "Area" →TAG
    4 3 / π →NUM * r 3 ^ * "Volume" →TAG
  »
»
```

Character Translations

When data is transferred between the HP 48 and a computer using translate code 2 (000→159) or 3 (000→255), conversions are used to represent some characters.

For data being transferred to a computer with translate codes 2 or 3, each \ is replaced with \\. For data being transferred to the HP 48, characters may be converted using a text conversion or \xxx, where xxx is the three-digit (decimal) character code.

The following table shows the text conversions for characters above code 127.

DEC	HEX	HP 48	ASCII	DEC	HEX	HP 48	ASCII
128	80	◀	\<)	148	94	¶	\Gn
129	81	⌘	\x-	149	95	⊖	\Gh
130	82	∇	\.V	150	96	λ	\Gl
131	83	√	\v/	151	97	ρ	\Gr
132	84	ƒ	\.S	152	98	σ	\Gs
133	85	Σ	\GS	153	99	τ	\Gt
134	86	►	\ >	154	9A	ω	\Gw
135	87	π	\pi	155	9B	△	\GD
136	88	ð	\.d	156	9C	Π	\PI
137	89	≤	\<=	157	9D	Ω	\GW
138	8A	≥	\>=	158	9E	▪	\ []
139	8B	≠	\=/	159	9F	∞	\oo
140	8C	α	\Ga	171	AB	⋄	\<<
141	8D	→	\->	176	B0	▪	\^o
142	8E	←	\<-	181	B5	μ	\Gm
143	8F	↓	\ v	187	BB	⊗	\>>
144	90	↑	\ ^	215	D7	×	\.x
145	91	γ	\Gg	216	D8	ø	\O/
146	92	δ	\Gd	223	DF	ß	\Gb
147	93	ε	\Ge	247	F7	÷	\:-

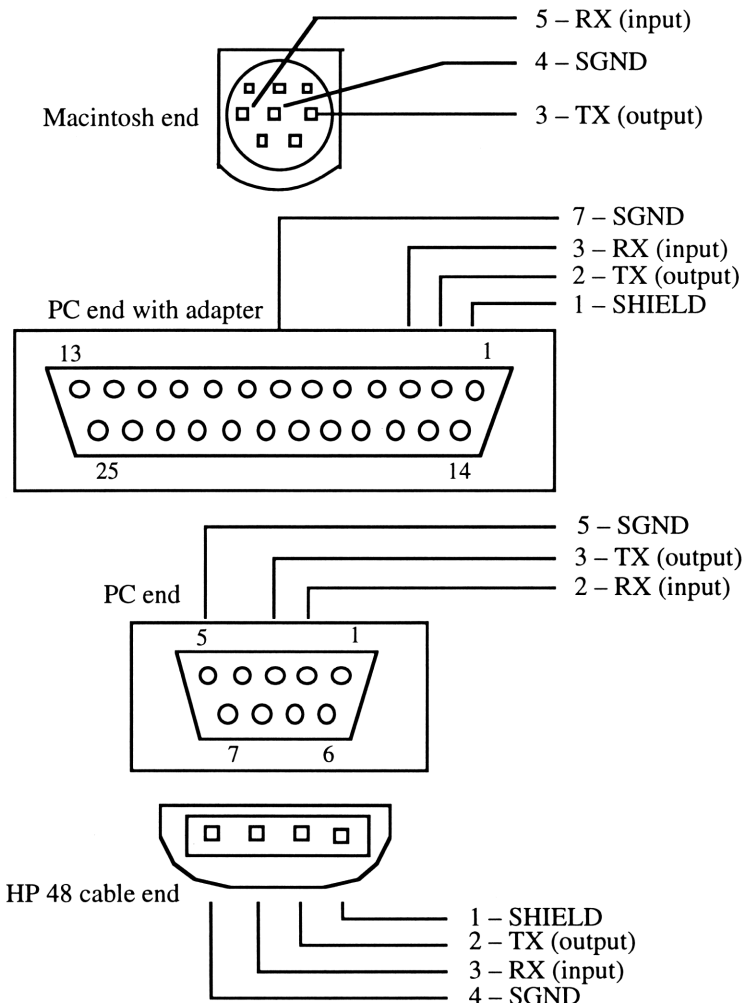
IOPAR

The reserved variable *IOPAR* may only reside in the HOME directory. Other variables of the same name in subdirectories will be ignored by the I/O commands.

IOPAR		
{ baud parity receive-pacing transmit-pacing checksum translate-code }		
Parameter	Description	Default
baud	1200, 2400*, 4800, or 9600	9600
parity	0=none*, 1=odd, 2=even, 3=mark, 4=space Negative parity value = transmit only	None
receive-pacing†	Value ≠ 0 sends XOFF if HP 48 buffer full	0
transmit-pacing†	Value ≠ 0 stops transmission if XOFF received	0
checksum	1=1 digit arithmetic, 2=2 digit arithmetic, 3=CRC	3
translate-code	0=none, 1=LF to CR-LF, 2=128-159, 3=128-255	1
*IR is 2400 baud, no parity only †Not used by Kermit		

Cables

Serial cables are available for a PC (HP F1015A) or an Apple Macintosh computer (HP F1016A). Connectivity kits include both a cable and data transfer software – HP F1201A for the PC, HP F1202A for the Macintosh.



Printer Control

The following system flags (default clear) control output to the printer as follows:

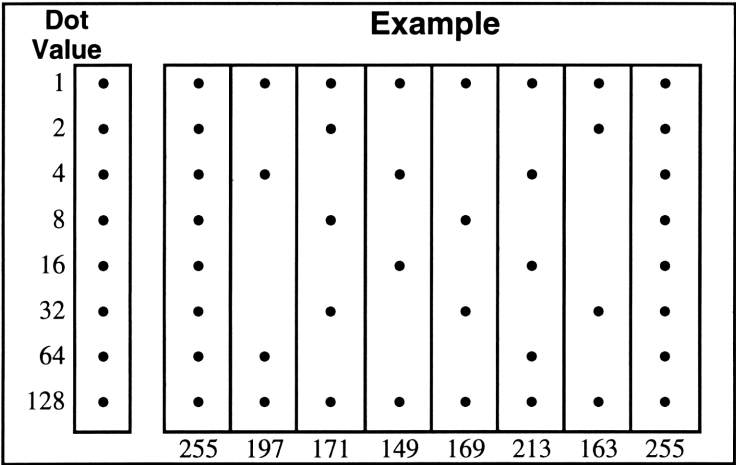
Flag	Clear	Set
-34	IR printer	Serial printer
-37	Single-spaced	Double-spaced
-38	Line feeds	No line feeds

The following control codes guide the operation of the HP 82240B printer:

Printer Command	Control Codes*
Carriage right	4
Carriage return/LF	10
Column graphics	27 n C ₁ ... C _n †
Roman 8 character set ‡	27 248
ISO 8859-1 character set	27 249
Underline off ‡	27 250
Underline on	27 251
Single wide print ‡	27 252
Double wide print	27 253
Self-test	27 254
Reset	27 255
*Decimal value †1<=n<=166 ‡Default mode	

Codes 248 and 249 were not included in the original HP 82240A printer. Characters 148 and 160 were blank on early versions of the HP 82240A printer. The HP 48 character set can be remapped to match the HP 82240A printer with the OLDPRT command.

Example: The program COL82 prints a simple graphics pattern on the HP 82240.



COL82 154.5 Bytes Checksum #E8EDh

```
«
27 8                               8-byte graphics command
255 197 171 149                   Graphics data
169 213 163 255
"" 1 10 START                     Loop start
SWAP CHR SWAP +                   Accumulate data
NEXT PR1                          Loop end, print graphics
DROP
»
```

PRTPAR

The reserved variable *PRTPAR* may only reside in the HOME directory. Other variables of the same name in subdirectories will be ignored by the print commands.

PRTPAR		
{ delay “remap” linelen “lineterm” }		
Parameter	Description	Default
delay	Time required to print line: $0 \leq t \leq 6.9$ seconds	1.8
“remap”	Character set remapping string	””
linelen	Serial print line length	80
“lineterm”	Serial print line terminating characters	“CR LF”

Messages

This chapter lists many of the messages built into the HP 48. Messages not listed are those that support the various input form user interfaces. The number ranges for those messages are listed at the end of this chapter.

A message number supplied to the command DOERR stops a program and displays the specified message number. The System-RPL object JstGETTHEMSG returns a message (see *System Programming*).

Hex	Dec	General Messages
001	1	Insufficient Memory
002	2	Directory Recursion
003	3	Undefined Local Name
004	4	Undefined XLIB Name
005	5	Memory Clear
006	6	Power Lost
007	7	Warning:
008	8	Invalid Card Data
009	9	Object In Use
00A	10	Port Not Available
00B	11	No Room in Port
00C	12	Object Not in Port
00D	13	Recovering Memory
00E	14	Try To Recover Memory?
00F	15	Replace RAM, Press ON
010	16	No Mem To Config All
101	257	No Room to Save Stack
102	258	Can't Edit Null Char.
103	259	Invalid User Function
104	260	No Current Equation
106	262	Invalid Syntax

Hex	Dec	Object Types
107	263	Real Number
108	264	Complex Number
109	265	String
10A	266	Real Array
10B	267	Complex Array
10C	268	List
10D	269	Global Name
10E	270	Local Name
10F	271	Program
110	272	Algebraic
111	273	Binary Integer
112	274	Graphic
113	275	Tagged
114	276	Unit
115	277	XLIB Name
116	278	Directory
117	279	Library
118	280	Backup
119	281	Function
11A	282	Command
11B	283	System Binary
11C	284	Long Real
11D	285	Long Complex
11E	286	Linked Array
11F	287	Character
120	288	Code
121	289	Library Data
122	290	External

Hex	Dec	General Messages
123	291	<i>Null message</i>
124	292	LAST STACK Disabled
125	293	LAST CMD Disabled
126	294	HALT Not Allowed
127	295	Array
128	296	Wrong Argument Count
129	297	Circular Reference
12A	298	Directory Not Allowed
12B	299	Non-Empty Directory
12C	300	Invalid Definition
12D	301	Missing Library
12E	302	Invalid PPAR
12F	303	Non-Real Result
130	304	Unable to Isolate

Hex	Dec	Low Memory
131	305	No Room to Show Stack
132	306	Warning
133	307	Error:
134	308	Purge?
135	309	Out of Memory
136	310	Stack
137	311	Last Stack
138	312	Last Commands
139	313	Key Assignments
13A	314	Alarms
13B	315	Last Arguments
13C	316	Name Conflict
13D	317	Command Line

Hex	Dec	Stack Operations
201	513	Too Few Arguments
202	514	Bad Argument Type
203	515	Bad Argument Value
204	516	Undefined Name
205	517	LASTARG Disabled

Hex	Dec	EquationWriter
206	518	Incomplete Subexpression
207	519	Implicit () off
208	520	Implicit () on

Hex	Dec	Floating Point Errors
301	769	Positive Underflow
302	770	Negative Underflow
303	771	Overflow
304	772	Undefined Result
305	773	Infinite Result

Hex	Dec	Array
501	1281	Invalid Dimension
502	1282	Invalid Array Element
503	1283	Deleting Row
504	1284	Deleting Column
505	1285	Inserting Row
506	1286	Inserting Column

Hex	Dec	Statistics
601	1537	Invalid Σ Data
602	1538	Nonexistent Σ DAT
603	1539	Insufficient Σ Data
604	1540	Invalid Σ PAR
605	1541	Invalid Σ Data LN(Neg)
606	1542	Invalid Σ Data LN(0)

Hex	Dec	Plot, Solve, Stat
607	1543	Invalid EQ
608	1544	Current equation:
609	1545	No current equation.
60A	1546	Enter eqn, press NEW
60B	1547	Name the equation, press ENTER
60C	1548	Select plot type
60D	1549	Empty catalog
60E	1550	undefined
60F	1551	No stat data to plot
610	1552	Autoscaling
611	1553	Solving for
612	1554	No current data. Enter
613	1555	data point, press $\Sigma+$
614	1556	Select a model

Hex	Dec	Alarms
615	1557	No alarms pending.
616	1558	Press ALRM to create
617	1559	Next alarm:
618	1560	Past due alarm:
619	1561	Acknowledged
61A	1562	Enter alarm, press SET
61B	1563	Select repeat interval

Hex	Dec	I/O, Plot, Solve, Stat
61C	1564	I/O setup menu
61D	1565	Plot type:
61E	1566	" "
61F	1567	(OFF SCREEN)
620	1568	Invalid PTYPE
621	1569	Name the stat data, press ENTER
622	1570	Enter value (zoom out if >1), press ENTER

Hex	Dec	I/O, Plot, Solve, Stat
623	1571	Copied to stack
624	1572	x axis zoom w/AUTO.
625	1573	x axis zoom.
626	1574	y axis zoom.
627	1575	x and y-axis zoom.
628	1576	IR/wire:
629	1577	ASCII/binary:
62A	1578	baud:
62B	1579	parity:
62C	1580	checksum type:
62D	1581	translate code:
62E	1582	Enter matrix, then NEW
A01	2561	Bad Guess(es)
A02	2562	Constant?
A03	2563	Interrupted
A04	2564	Root
A05	2565	Sign Reversal
A06	2566	Extremum
A07	2567	Left
A08	2568	Right
A09	2569	Expr

Hex	Dec	Unit Management
B01	2817	Invalid Unit
B02	2818	Inconsistent Units

Hex	Dec	I/O and Printing
C01	3073	Bad Packet Block Check
C02	3074	Timeout
C03	3075	Receive Error
C04	3076	Receive Buffer Overrun
C05	3077	Parity Error
C06	3078	Transfer Failed
C07	3079	Protocol Error
C08	3080	Invalid Server Cmd.
C09	3081	Port Closed
C0A	3082	Connecting
C0B	3083	Retry #
C0C	3084	Awaiting Server Cmd.
C0D	3085	Sending
C0E	3086	Receiving
C0F	3087	Object Discarded
C10	3088	Packet #
C11	3089	Processing Command
C12	3090	Invalid IOPAR
C13	3091	Invalid PRTPAR
C14	3092	Low Battery
C15	3093	Empty Stack
C16	3094	Row
C17	3095	Invalid Name

Hex	Dec	Time
D01	3329	Invalid Date
D02	3330	Invalid Time
D03	3331	Invalid Repeat
D04	3332	Nonexistent Alarm

Hex	Dec	Polynomial Root Finder
C001	49153	Unable to find root

Hex	Dec	Multiple Equation Solver
E401	58369	Invalid Mpar
E402	58370	Single Equation
E403	58371	EQ Invalid for MINIT
E404	58372	Too Many Unknowns
E405	58373	All Variables Known
E406	58374	Illegal During MROOT
E407	58375	Solving for
E408	58376	Searching

Start	End	Unlisted Message Numbers
B901	B99B	Miscellaneous
BA01	BA43	I/O operations
BB01	BB3F	Statistics
BC01	BC3B	Time system
BD01	BD27	Symbolic operations
BE01	BE77	Plotting
BF01	BF56	Solver
E101	E129	Constants Library
E301	E304	Equation Library
E701	E708	Minehunt game

Menus

Custom Menu

A custom menu may be created using a list of objects supplied to the MENU or TMENU commands.

{ *Key*₁ *Key*₂ *Key*₃ ... }

The objects that define each key in the menu may range in complexity from a real number to a list definition with a graphics object for the menu key label and separate actions for the primary and left- or right-shifted planes.

The Variable CST. The MENU command stores the definition in the reserved variable *CST* and immediately displays the menu. Each directory may have a different variable *CST*. A name may be stored in *CST* which references a variable containing the menu definition. The TMENU command does not affect *CST*.

Menu Contents. Menus may contain any object, but the functionality of the key is determined by the type of the object:

- Names work the same way as the VAR menu.
- Keys with string definitions echo the string.
- Directory names change to the directory.
- Unit objects act as unit catalog entries:
 - Primary keys append the unit on the key to the numerator of the level 1 object.
 - Left-shifted keys convert the level 1 object to the unit on the key.
 - Right-shifted keys append the unit on the key to the denominator of the level 1 object.
- Backup objects act like the port 0, 1, and 2 menus.
- Labeled objects can be used to identify menu key actions and can provide optional shifted functionality.

Labels. A menu key can have a label that is different than its key action. The most versatile key definition provides separate objects for the label, primary, left-shifted, and right-shifted actions. Either a string or a graphics object 8 rows high by 21 columns wide may be supplied as the label.

Example: The following list contains a menu definition for six keys: a variable, string, unit object, labelled program, a definition that uses a graphics object for the menu label, and labelled key definition with shifted functionality:

MENUEX 226.5 Bytes Checksum #C051h





















```
{
  X
  "HELLO"
  1_m^3
  { "PRG" « 2 * 3 + » }
  {
    GROB 21 8 0000000404000A0A0005151080A020FFFFFF100F100004000
    "Kilroy was here!"
  }
  { "CPL"
    {
      « CPL »           primary action
      « 'CPL' STO »     left-shifted action
      « 'CPL' RCL »     right-shifted action
    }
  }
}
```








































































Two more extensive examples of custom menus can be found in the *Customizing the Solver* example in *Example Programs*.

Built-In Menus

The commands MENU, TMENU, and RCLMENU store and recall menu numbers in the form *mm.pp*, where *mm* is the menu number and *pp* is the page number. The first two tables list menu numbers for the HP 48G/GX, the third table lists menu numbers for the HP 48S/SX.

HP 48G/GX Menu Numbers			
#	Menu Name	#	Menu Name
0	LAST Menu	30	 SOLVE ROOT SOLVR
1	CST	31	PRG WHILE
2	VAR	32	PRG TEST
3	MTH	33	PRG TYPE
4	MTH VECTR	34	PRG LIST
5	MTH MATR	35	PRG LIST ELEM
6	MTH MATR MAKE	36	PRG LIST PROC
7	MTH MATR NORM	37	PRG GROB
8	MTH MATR FACTR	38	PRG PICT
9	MTH MATR COL	39	PRG IN
10	MTH MATR ROW	40	PRG OUT
11	MTH LIST	41	PRG RUN
12	MTH HYP	42	 UNITS
13	MTH PROB	43	 UNITS LENG
14	MTH REAL	44	 UNITS AREA
15	MTH BASE	45	 UNITS VOL
16	MTH BASE LOGIC	46	 UNITS TIME
17	MTH BASE BIT	47	 UNITS SPEED
18	MTH BASE BYTE	48	 UNITS MASS
19	MTH FFT	49	 UNITS FORCE
20	MTH CMPL	50	 UNITS ENRG
21	MTH CONS	51	 UNITS POWR
22	PRG	52	 UNITS PRESS
23	PRG BRCH	53	 UNITS TEMP
24	PRG IF	54	 UNITS ELEC
25	PRG CASE	55	 UNITS ANGL
26	PRG START	56	 UNITS LIGHT
27	PRG FOR	57	 UNITS RAD
28	 EDIT	58	 UNITS VISC
29	PRG DO	59	 UNITS

HP 48G/GX Menu Numbers (cont.)			
#	Menu Name	#	Menu Name
60	PRG ERROR IFERR	89	 PLOT STAT Σ PAR
61	PRG ERROR	90	 PLOT STAT Σ PAR MODL
62	 CHARS	91	 PLOT STAT DATA
63	 MODES	92	 PLOT FLAG
64	 MODES FMT	93	 SYMBOLIC
65	 MODES ANGL	94	 TIME
66	 MODES FLAG	95	 TIME ALARM
67	 MODES KEYS	96	 STAT
68	 MODES MENU	97	 STAT DATA
69	 MODES MISC	98	 STAT Σ PAR
70	 MEMORY	99	 STAT Σ PAR MODL
71	 MEMORY DIR	100	 STAT IVAR
72	 MEMORY ARITH	101	 STAT PLOT
73	 STACK	102	 STAT FIT
74	 SOLVE	103	 STAT SUMS
75	 SOLVE ROOT	104	 IO
76	 SOLVE DIFFEQ	105	 IO SRVR
77	 SOLVE POLY	106	 IO IOPAR
78	 SOLVE SYS	107	 IO PRINT
79	 SOLVE TVM	108	 IO PRINT PRTPA
80	 SOLVE TVM SOLVR	109	 IO SERIA
81	 PLOT	110	 LIBRARY
82	 PLOT PTYPE	111	 LIBRARY PORTS
83	 PLOT PPAR	112	 LIBRARY
84	 PLOT 3D	113	 EQ LIB
85	 PLOT 3D PTYPE	114	 EQ LIB EQLIB
86	 PLOT 3D VPAR	115	 EQ LIB COLIB
87	 PLOT STAT	116	 EQ LIB MES
88	 PLOT STAT PTYPE	117	 EQ LIB UTILS

HP 48S/SX Menu Numbers			
#	Menu Name	#	Menu Name
0	LAST Menu	30	SOLVE SOLVR
1	CST	31	 PLOT
2	VAR	32	PLOT PTYPE
3	MTH	33	PLOT PLOTR
4	MTH PARTS	34	 ALGEBRA
5	MTH PROB	35	 TIME
6	MTH HYP	36	TIME ADJUST
7	MTH MATRX	37	TIME ALRM
8	MTH VECTR	38	TIME ALRM RPT
9	MTH BASE	39	TIME SET
10	PRG	40	 STAT
11	PRG STK	41	STAT MODL
12	PRG OBJ	42	 UNITS
13	PRG DISP	43	UNITS LENG
14	PRG CTRL	44	UNITS AREA
15	PRG BRCH	45	UNITS VOL
16	PRG TEST	46	UNITS TIME
17	PRINT	47	UNITS SPEED
18	I/O	48	UNITS MASS
19	I/O SETUP	49	UNITS FORCE
20	 MODES	50	UNITS ENRG
21	 MODES	51	UNITS POWR
22	 MEMORY	52	UNITS PRESS
23	 MEMORY	53	UNITS TEMP
24	 LIBRARY	54	UNITS ELEC
25	LIBRARY PORT 0	55	UNITS ANGL
26	LIBRARY PORT 1	56	UNITS LIGHT
27	LIBRARY PORT 2	57	UNITS RAD
28	 EDIT	58	UNITS VISC
29	 SOLVE	59	 UNITS

User-Defined Keys

Variables, programs, commands, or strings may be assigned to any key on the HP 48. When 1-User or User mode is active, these objects are evaluated in place of the standard key definitions.

The ASN and STOKEYS commands may be used to assign an object to a key. The command RCLKEYS recalls the current key assignments, and DELKEYS deletes one or more assignments. These commands are shown on the next page.

Setting User Mode

1-User mode may be set by pressing $\boxed{\leftarrow}\boxed{\text{USR}}$. 1-User mode remains in effect for only one operation. User mode may be locked by pressing $\boxed{\leftarrow}\boxed{\text{USR}}$ twice or by setting flag -62. When flag -61 is set $\boxed{\leftarrow}\boxed{\text{USR}}$ toggles user mode, and 1-User mode is not available.

Key Locations

The notation $rc.p$ specifies the location of a key where r is the row, c is the column, and p is the plane.

p	Primary Planes	p	Alpha Planes
0 or 1	Unshifted	4	Alpha
2	Left-shifted	5	Alpha left-shifted
3	Right-shifted	6	Alpha right-shifted

Examples: the $\boxed{\text{ENTER}}$ key is 51 (or 51.1), the left-shifted $\boxed{6}$ key is 74.2, and the alpha right-shifted $\boxed{\text{CST}}$ is 23.6.

Standard Keys

When User mode is set, the standard key definitions apply to all keys which have not been reassigned. The standard key definitions may be disabled by supplying the S parameter to the DELKEYS command. The symbol S refers to standard key definitions. An individual standard key definition may be reactivated by supplying SKEY as the assigned object for ASN. All standard keys may be reactivated by supplying SKEY to STOKEYS.

Flags



User flags are numbered 1 through 64. System flags are numbered from -1 through -64. By convention, application developers are encouraged to restrict their use of user flags to the range 31–64.

All flags are clear by default, except for the wordsize (flags -5 to -10).

The related commands SF, CF, FS?, FC?, FS?C, and FC?C are found in the **PRG TEST** menu. RCLF and STOF return or store a list of two binary integers representing the system and user flag sets (remember to use a wordsize of 64).

Flag	Description	Clear	Set	Default
Symbolic Math				
-1	Principal Solution	General solutions	Principal solutions	Clear
-2	Symbolic Constants	Symbolic form	Numeric form	Clear
-3	Numeric Results	Symbolic results	Numeric results	Clear
-4	Not used			
Binary Integer Math				
-5	Binary integer wordsize $n + 1: 0 \leq n \leq 63$			64
-10	Flag -10 is the most significant bit			
	Base	-11	-12	DEC
-11	DEC	Clear	Clear	
and	BIN	Clear	Set	
-12	OCT	Set	Clear	
	HEX	Set	Set	
-13	Not used			
Finance				
-14	TVM Payment Mode	End of Period	Beginning of Period	End
Coordinate System		-15	-16	Rect.
-15	Rectangular	Clear	Clear	
and	Cylindrical Polar	Clear	Set	
-16	Spherical Polar	Set	Set	
Trigonometric Mode		-17	-18	Degrees
-17	Degrees	Clear	Clear	
and	Radians	Set	Clear	
-18	Grads	Clear	Set	

Flag	Description	Clear	Set	Default
Math Exception				
-19	Vector/complex	Vector	Complex	Vector
-20	Underflow Exception	Return 0, set -23 or -24	Error	Clear
-21	Overflow Exception	Return \pm MAXR, set -25	Error	Clear
-22	Infinite Result	Error	Return \pm MAXR, set -26	Error
-23	Neg. Underflow Ind.	No Exception	Exception	Clear
-24	Pos. Underflow Ind.	No Exception	Exception	Clear
-25	Overflow Indicator	No Exception	Exception	Clear
-26	Infinite Result Ind.	No Exception	Exception	Clear
-27	Symbolic Decompilation	'X+Y*i' \rightarrow '(X,Y)'	'X+Y*i' \rightarrow 'X+Y*i'	Clear
Plotting and Graphics				
-28	Plotting Multiple Functions	Plotted serially	Plotted simultaneously	Clear
-29	Trace mode	Trace off	Trace on	Off
-30	Not used			
-31	Curve Filling	Filling Enabled	Filling Disabled	Enabled
-32	Graphics Cursor	Visible light bkgnd	Visible dark bkgnd	Light
I/O and Printing				
-33	I/O Device	Wire	IR	Wire
-34	Printing Device	IR	Wire	IR
-35	I/O Data Format	ASCII	Binary	ASCII
-36	RECV Overwrite	New variable	Overwrite	New
-37	Double-spaced Print	Single	Double	Single
-38	Linefeed	Inserts LF	Suppresses LF	Inserts
-39	Kermit Messages	Msg displayed	Msg suppressed	Clear
Time Management				
-40	Clock Display	No clock display	All times	Clear
-41	Clock Format	12 hour	24 hour	12 hour
-42	Date Format	MM/DD/YY	DD.MM.YY	Clear
-43	Rpt. Alarm Resched.	Rescheduled	Not rescheduled	Clear
-44	Acknowledged Alarms	Deleted	Saved	Deleted
Notes: If flag -43 is set, unacknowledged repeat alarms are <i>not</i> rescheduled. If flag -44 is set, acknowledged alarms are saved in the alarm catalog.				

Flag	Description	Clear	Set	Default
Display Format				
-45→ -48	Set the number of digits in Fix, Scientific, and Engineering modes			0
Number Display Format		-49	-50	STD
-49 and -50	STD FIX SCI ENG	Clear Clear Set Set	Clear Set Clear Set	
-51 -52 -53	Fraction Mark Single Line Display Precedence	Decimal Multi-line () suppressed	Comma Single-line () displayed	Decimal Multi Clear
Miscellaneous				
-54	Tiny Array Elements	Replaces "tiny" pivots with 0	No replacement	Replaces
-55	Last Arguments	Saved	Not saved	Saved
-56	Beep	On	Off	On
-57	Alarm Beep	On	Off	On
-58	Verbose Messages	On	Off	On
-59	Fast Catalog Display	Off	On	Off
-60	Alpha Key Action	Twice to lock	Once to lock	Twice
-61	USR Key Action	Twice to lock	Once to lock	Twice
-62	User Mode	Not Active	Active	Clear
-63	Vectored Enter	Off	On	Off
-64	Set by GETI or PUTI when their element indices wrap around			
Equation Library				
60	Units Type	SI units	English units	SI
61	Units Usage	Units used	Units not used	Used
Multiple Equation Solver				
63	Variable State Change	 recalls variable	 toggles variable state	Recalls

Built-In Units

The prefixes y, Y, z, and Z are new to the HP 48G/GX.

UNIT PREFIXES			
HP 48 Symbol	Prefix	Number	Name
Y	yotta	+24	quintillion quadrillion trillion billion million thousand hundred ten tenth hundredth thousandth millionth billionth trillionth quadrillionth quintillionth
Z	zetta	+21	
E	exa	+18	
P	peta	+15	
T	tera	+12	
G	giga	+9	
M	mega	+6	
k, K	kilo	+3	
h, H	hecto	+2	
D	deka	+1	
d	deci	-1	
c	centi	-2	
m	milli	-3	
μ	micro	-6	
n	nano	-9	
p	pico	-12	
f	femto	-15	
a	atto	-18	
z	zepto	-21	
y	yocto	-24	

Prefix-unit combinations that match built-in units are: au, cd, ct, cu, ft, flam, kph, mph, min, nmi, Pa, ph, pt, yd, and yr.

Unit	Name	Type	Value
a	Are	area	100 m ²
A	Ampere	electric current	1 A
Å	Angstrom	length	1x10 ⁻¹⁰ m
acre	Acre	area	4046.87260987 m ²
arcmin	Minute of arc	planar angle	4.62962962963x10 ⁻⁵
arcs	Second of arc	planar angle	.71604938272x10 ⁻⁷
atm	Atmosphere	pressure	101325 kg/m•s ²
au	Astronomical unit	length	1.495979x10 ¹¹ m
b	Barn	area	1x10 ⁻²⁸ m ²
bar	Bar	pressure	100000 kg/m•s ²
bbl	Barrel	volume	.158987294928 m ³
Bq	Becquerel	activity	1/s
Btu	Int'l Table Btu	energy	105505585262 kg•m ² /s ²
bu	Bushel	volume	.03523907 m ³
c	Speed of light	speed	299792458 m/s
C	Coulomb	electric charge	1 A•s
°C	Degree Celsius	temperature	
cal	Calorie	energy	4.1868 kg•m ² /s ²
cd	Candela	luminous intensity	1 cd
chain	Chain	length	20.1168402337 m
Ci	Curie	activity	3.7x10 ¹⁰ 1/s
cm	Centimeter	length	.01 m
cm^2	Square centimeter	area	.0001 m ²
cm^3	Cubic centimeter	volume	.000001 m ³
cm/s	Centimeter per second	speed	.01 m/s
ct	Carat	mass	.0002 kg
cu	U.S. cup	volume	.0002365882365 m ³
d	Day	time	86400 s
dyn	Dyne	force	.00001 kg•m/s ²
erg	Erg	energy	.0000001 kg•m ² /s ²
eV	Electron volt	energy	1.60219x10 ⁻¹⁹ kg•m ² /s ²
F	Farad	capacitance	1 A ² •s ⁴ /kg•m ²
°F	Degree Fahrenheit	temperature	
fath	Fathom	length	1.82880365761 m
fbm	Board foot	volume	.002359737216 m ³
fc	Footcandle	illuminance	10.7639104167 cd•sr/m ²
Fdy	Faraday	electric charge	96487 A•s
fermi	Fermi	length	1x10 ⁻¹⁵ m
flam	Footlambert	luminance	3.42625909964 cd/m ²

Unit	Name	Type	Value
ft	Int'l foot	length	.3048 m
ft^2	Square foot	area	.09290304 m ²
ft^3	Cubic foot	volume	.028316846592 m ³
ftUS	U.S. survey foot	length	.304800609601 m
ft/s	Feet/second	speed	.3048 m/s
ft*lb _f	Foot-pound-force	energy	1.35581794833 kg•m ² /s ²
g	Gram	mass	.001 kg
ga	Standard freefall	acceleration	9.80665 m/s ²
gal	U.S. gallon	volume	.003785411784 m ³
galC	Canadian gallon	volume	.00454609 m ³
galUK	U.K. gallon	volume	.004546092 m ³
gf	Gram-force	force	.00980665 kg•m/s ²
grad	Grade	planar angle	.0025
grain	Grain	mass	.00006479891 kg
Gy	Gray	absorbed dose	1 m ² /s ²
h	Hour	time	3600 s
H	Henry	inductance	1 kg•m ² /A ² •s ²
ha	Hectare	area	10000 m ²
hp	Horsepower	power	745.699871582 kg•m ² /s ³
Hz	Hertz	frequency	1/s
in	Inch	length	.0254 m
in^2	Square inch	area	.00064516 m ²
in^3	Cubic inch	volume	.000016387064 m ³
inHg	Inch of mercury	pressure	3386.38815789 kg/m•s ²
inH2O	Inch of water	pressure	248.84 kg/m•s ²
J	Joule	energy	1 kg•m ² /s ²
K	Kelvin	temperature	1 K
kcal	Kilocalorie	energy	4186 kg•m ² /s ²
kg	Kilogram	mass	1 kg
kip	Kilopound-force	force	4448.22161526 kg•m/s ²
km	Kilometer	length	1 km
km^2	Square kilometer	area	1 km ²
knot	Nautical mile per hour	speed	.514444444444 m/s
kph	Kilometer per hour	speed	.277777777778 m/s
l	Liter	volume	.001 m ³
lam	Lambert	luminance	3183.09886184 cd/m ²
lb	Avoirdupois pound	mass	.45359237 kg
lb _f	Pound-force	force	4.44822161526 kg•m/s ²
lbt	Troy pound	mass	.3732417216 kg
lm	Lumen	luminous flux	1 cd•sr
lx	Lux	illuminance	1 cd•sr/m ²
lyr	Light year	length	9.46052840488x10 ¹⁵ m

Unit	Name	Type	Value
m	Meter	length	1 m
m ²	Square meter	area	1 m ²
m ³	Cubic meter	volume	1 m ³
μ	Micron	length	.000001 m
MeV	Mega electron volt	energy	1.60219×10 ⁻¹³ kg•m ³ /s ²
mho	Mho	electric conductance	1 A ² •s ³ /kg•m ²
mi	Int'l mile	length	1609.344 m
mi ²	Int'l square mile	area	2589988.11034 m ²
mil	Mil	length	.0000254 m
min	Minute	time	60 s
miUS	U.S. statute mile	length	1609.34721869 m
miUS ²	U.S. statute sq. mile	area	258998.47032 m ²
mm	Millimeter	length	.001 m
mmHG	Millimeter of mercury	pressure	133.322368421 kg/m•s ²
ml	Milliliter	volume	.000001 m ³
mol	Mole	amount of substance	1 mol
Mpc	Megaparsec	length	3.08567818585×10 ²² m
mph	Mile per hour	speed	.44704 m/s
m/s	Meter per second	speed	1 m/s
N	Newton	force	1 kg•m/s ²
nml	Nautical mile	length	1852 m
oz	Ounce	mass	.028349523125 kg
ozfl	U.S. fluid ounce	volume	2.95735295625×10 ⁻⁵ m ³
ozt	Troy ounce	mass	.0311034768 kg
ozUK	U.K. fluid ounce	volume	2.8413075×10 ⁻⁵ m ³
P	Poise	dynamic viscosity	.1 kg/m•s
Pa	Pascal	pressure	1 kg/m•s ²
pc	Parsec	length	3.08567818585×10 ¹⁶ m
pdl	Poundal	force	.138254954376 kg•m/s ²
ph	Phot	illuminance	10000 cd•sr/m ²
pk	Peck	volume	.0088097675 m ³
psi	Pound per sq. inch	pressure	6894.75729317 kg/m•s ²
pt	Pint	volume	.000473176473 m ³
qt	Quart	volume	.000946352946 m ³
r	Radian	planar angle	.1591549343092
R	Roentgen	radiation exposure	.000258 A•s/kg
°R	Degree Rankine	temperature	
rad	Rad	absorbed dose	.01 m ² /s ²
rd	Rod	length	5.02921005842 m
rem	Rem	dose equivalent	.01 m ² /s ²

Unit	Name	Type	Value
s	Second	time	1 s
S	Siemens	electric conductance	$1 \text{ A}^2 \cdot \text{s}^3 / \text{kg} \cdot \text{m}^2$
sb	Stilb	luminance	$10000 \text{ cd} / \text{m}^2$
slug	Slug	mass	14.5939029372 kg
sr	Steradian	solid angle	$.0795774715459$
st	Stere	volume	1 m^3
St	Stoke	kinematic viscosity	$.0001 \text{ m}^2 / \text{s}$
Sv	Sievert	dose equivalent	$.01 \text{ m}^2 / \text{s}^2$
t	Metric ton	mass	1000 kg
T	Tesla	magnetic flux	$1 \text{ kg} / \text{A} \cdot \text{s}^2$
tbsp	Tablespoon	volume	$1.47867647813 \times 10^{-5} \text{ m}^3$
therm	EEC therm	energy	$1055060000 \text{ kg} \cdot \text{m}^2 / \text{s}^2$
ton	Short ton	mass	907.18474 kg
tonUK	Long (U.K.) ton	mass	1016.0469088 kg
torr	Torr	pressure	$133.322368421 \text{ kg} / \text{m} \cdot \text{s}^2$
tsp	Teaspoon	volume	$4.92892159375 \times 10^{-6} \text{ m}^3$
u	Unified atomic mass	mass	$1.6605402 \times 10^{-27} \text{ kg}$
V	Volt	electrical potential	$1 \text{ kg} \cdot \text{m}^2 / \text{A} \cdot \text{s}^3$
W	Watt	power	$1 \text{ kg} \cdot \text{m}^2 / \text{s}^3$
Wb	Weber	magnetic flux	$1 \text{ kg} \cdot \text{m}^2 / \text{A} \cdot \text{s}^2$
yd	Int'l yard	length	$.9144 \text{ m}$
yd^2	Square yard	area	$.83612736 \text{ m}^2$
yd^3	Cubic yard	volume	$.764554857984 \text{ m}^3$
yr	Year	time	31556925.9747 s
°	Degree	planar angle	$2.77777777778 \times 10^{-3}$
Ω	Ohm	electric resistance	$1 \text{ kg} \cdot \text{m}^2 / \text{A}^2 \cdot \text{s}^3$

Equation Library Reference

Equation Reference

The command SOLVEQN may be used to place a set of equations from the Equation Library into the built-in Solver for single equations or the Multiple Equation Solver for multiple equation sets. The level 3 and 2 parameters specify the subject and title number. If the level 1 parameter is non-zero, the picture associated with the equation set will be placed in *PICT*.

SOLVEQN

Command

Places Equation Library equation(s) in solver

subject_number title_number *PICT_option* →

The following table shows the subject and title numbers that may be used with the SOLVEQN command. If the *TYPE* is listed as *S*, the title contains a single equation; *M* indicates a set of multiple equations. A *Y* listed under *PICTURE* indicates that a picture is associated with the title.

1	COLUMNS AND BEAMS		
<i>TITLE#</i>	<i>TITLE</i>	<i>TYPE</i>	<i>PICTURE</i>
1	Elastic Buckling	M	Y
2	Eccentric Columns	M	Y
3	Simple Deflection	S	Y
4	Simple Slope	S	Y
5	Simple Moment	S	Y
6	Simple Shear	S	Y
7	Cantilever Deflection	S	Y
8	Cantilever Slope	S	Y
9	Cantilever Moment	S	Y
10	Cantilever Shear	S	Y

2	ELECTRICITY		
<i>TITLE#</i>	<i>TITLE</i>	<i>TYPE</i>	<i>PICTURE</i>
1	Coulomb's Law	S	
2	Ohm's Law and Power	M	
3	Voltage Divider	S	Y
4	Current Divider	S	Y
5	Wire Resistance	S	
6	Series and Parallel R	M	Y
7	Series and Parallel C	M	Y
8	Series and Parallel L	M	Y
9	Capacitive Energy	S	
10	Inductive Energy	S	
11	RLC Current Delay	M	Y
12	DC Capacitor Current	M	
13	Capacitor Charge	S	
14	DC Inductor Voltage	M	
15	RC Transient	S	Y
16	RL Transient	S	Y
17	Resonant Frequency	M	
18	Plate Capacitor	S	Y
19	Cylindrical Capacitor	S	Y
20	Solenoid Inductance	S	Y
21	Toroid Inductance	S	Y
22	Sinusoidal Voltage	M	
23	Sinusoidal Current	M	
3	FLUIDS		
1	Pressure at Depth	S	Y
2	Bernoulli Equation	M	Y
3	Flow with Losses	M	Y
4	Flow in Full Pipes	M	Y

4	FORCES AND ENERGY		
TITLE#	TITLE	TYPE	PICTURE
1	Linear Mechanics	M	Y
2	Angular Mechanics	M	
3	Centripetal Force	M	
4	Hooke's Law	M	
5	1D Elastic Collisions	M	
6	Drag Force	S	
7	Law of Gravitation	S	
8	Mass-Energy Relation	S	
5	GASES		
1	Ideal Gas Law	M	Y
2	Ideal Gas State Chg	S	
3	Isothermal Expansion	M	
4	Polytropic Processes	M	
5	Isentropic Flow	M	
6	Real Gas Law	M	
7	Real Gas State Change	S	
8	Kinetic Theory	M	
6	HEAT TRANSFER		
1	Heat Capacity	M	Y
2	Thermal Expansion	M	
3	Conduction	M	
4	Convection	M	
5	Conduction+Convection	M	
6	Black Body Radiation	M	
7	MAGNETISM		
1	Straight Wire	S	Y
2	Force Between Wires	S	Y
3	B Field in Solenoid	S	Y
4	B Field in Toroid	S	Y

8	MOTION		
TITLE#	TITLE	TYPE	PICTURE
1	Linear Motion	M	Y
2	Object in Free Fall	M	
3	Projectile Motion	M	
4	Angular Motion	M	
5	Circular Motion	M	
6	Terminal Velocity	S	
7	Escape Velocity	S	
9	OPTICS		
1	Law of Refraction	S	Y
2	Critical Angle	S	Y
3	Brewster's Law	M	Y
4	Spherical Reflection	M	Y
5	Spherical Refraction	S	Y
6	Thin Lens	M	Y
10	OSCILLATIONS		
1	Mass-Spring System	M	Y
2	Simple Pendulum	M	Y
3	Conical Pendulum	M	Y
4	Torsional Pendulum	M	Y
5	Simple Harmonic	M	
11	PLANE GEOMETRY		
1	Circle	M	Y
2	Ellipse	M	Y
3	Rectangle	M	Y
4	Regular Polygon	M	Y
5	Circular Ring	M	Y
6	Triangle	M	Y

12	SOLID GEOMETRY		
<i>TITLE#</i>	<i>TITLE</i>	<i>TYPE</i>	<i>PICTURE</i>
1	Cone	M	Y
2	Cylinder	M	Y
3	Parallelepiped	M	Y
4	Sphere	M	Y
13	SOLID STATE DEVICES		
1	PN Step Junctions	M	Y
2	NMOS Transistors	M	Y
3	Bipolar Transistors	M	Y
4	JFETs	M	Y
14	STRESS ANALYSIS		
1	Normal Stress	M	Y
2	Shear Stress	M	Y
3	Stress on an Element	M	Y
4	Mohr's Circle	M	Y
15	WAVES		
1	Transverse Waves	M	
2	Longitudinal Waves	M	
3	Sound Waves	M	

Constants Reference

Name	Description
N_A	Avogadro's number
k	Boltzmann constant
V_m	Molar volume
R	Universal gas constant
$StdT$	Standard temperature
$StdP$	Standard pressure
σ	Stefan-Boltzmann constant
c	Speed of light in vacuum
ϵ_0	Permittivity of vacuum
μ_0	Permeability of vacuum
g	Acceleration due to gravity
G	Gravitational constant
h	Planck's constant
\hbar	Dirac's constant
q	Electronic charge
m_e	Electron rest mass
q/m_e	q/m_e ratio (electron charge-to-mass)
m_p	Proton rest mass
m_p/m_e	m_p/m_e ratio (proton, electron mass)
α	Fine structure constant
Φ_0	Magnetic flux quantum
F	Faraday constant
R_∞	Rydberg constant
a_0	Bohr radius
μ_B	Bohr magneton

Name	Description
μN	Nuclear magneton
λ_0	Photon wavelength
f_0	Photon frequency
λ_c	Compton wavelength
rad	1 radian
2π	2π radians
angl	180° angle (in current trig mode if no units)
c3	Wien's displacement law constant
kq	k/q (Boltzmann, electronic charge)
ϵ_0/q	ϵ_0/q (permittivity, electronic charge)
$q\epsilon_0$	$q\epsilon_0$ (electronic charge, permittivity)
ϵ_{si}	Dielectric constant of silicon
ϵ_{ox}	Dielectric constant of silicon dioxide
I0	Reference intensity

Command Index

This index lists the commands and functions in the HP 48, grouped into subject areas. Some commands or functions appear more than once.

BINARY INTEGER MATH	
AND	Logical bit-by-bit AND
ASR	Arithmetic shift right
B→R	Binary-to-real conversion
NOT	One's complement
OR	Logical bit-by-bit OR
RCWS	Recalls the binary integer wordsize
RL	Rotates left by one bit
RLB	Rotates left by one byte
RR	Rotates right by one bit
RRB	Rotates right by one byte
R→B	Real-to-binary conversion
SL	Shifts left by one bit
SLB	Shifts left by one byte
SR	Shifts right by one bit
SRB	Shifts right by one byte
STWS	Sets the binary integer wordsize
XOR	Logical bit-by-bit XOR

COMPLEX NUMBER OPERATIONS	
ABS	$\text{SQRT}(x^2+y^2)$
ARG	Returns the polar angle θ of a coordinate pair (x,y)
CONJ	Complex conjugate
C→R	Complex-to-real conversion
i	Symbolic constant i
IM	Returns imaginary part of a number or array
NEG	Negates an argument
OBJ→	Complex decomposition
RE	Returns the real part of a complex number
R→C	Real-to-complex conversion
SIGN	Returns unit vector in the direction of the argument
V→	Separates (x,y) into x and y or r and θ
→V2	Combines x and y into (x,y) or (r, θ) if flag -19 is set

ARRAY AND LIST OPERATIONS

ADD	Same as +, but performs element-wise addition of objects in lists
ARRAY→	Separate array into individual elements
→ARRAY	Combines numbers into an array
CONVERT	Performs a unit conversion
DOLIST	Applies n arguments in list to an object
DOSUBS	Executes a program or command using arguments in a list
DTAG	Removes all tags from object
ENDSUB	Returns the maximum frame number for DOSUBS
EQ→	Separates equation into left and right sides
GET	Gets an element from a list or array
GETI	Gets an element from a list or array, increments and returns the index, and returns the list or array
HEAD	Returns the first object in a list
LIST→	Separates a list into individual objects
→LIST	Combines objects into a list
ΔLIST	Computes first differences of elements in a list
ΣLIST	Sums the elements in a list
ΠLIST	Returns the product of elements in a list
NSUB	Returns the current frame number for DOSUBS
OBJ→	Decomposes a composite object into individual components.
POS	Finds an object in a list
PUT	Replaces an element in a list or array
PUTI	Replaces an element in an array or list, increments and returns the index, and returns the list or array
REPL	Writes an object into another object
REVLIST	Reverses the the order of elements in a list
SEQ	Generates a list of results from repeated execution of an object
SIZE	Finds the number of elements in a list
SORT	Sorts elements in lists
STREAM	Cumulatively applies object to arguments in a list
SUB	Extracts a portion of a list
→TAG	Builds a tagged object
TAIL	Returns a list less its first object
→V2	Combines two real numbers into a vector
→V3	Combines three real numbers into a vector
V→	Separates a 2- or 3-element vector

CONSTANTS

CONLIB	Displays the Constants Library catalog
CONST	Returns a constant from the Constants Library
i	Symbolic constant i
e	Symbolic constant e
MAXR	Symbolic constant – maximum HP 48 real number
MINR	Symbolic constant – minimum HP 48 real number
π	Symbolic constant π

CUSTOMIZATION	
ASN	Make a single user-key assignment
DELKEYS	Clears user-key assignments
DEFINE	Creates variable or user-defined function
MENU	Selects a built-in menu or creates a custom menu
ORDER	Rearranges the VAR menu
RCLF	Returns a list containing the system and user flags
RCLKEYS	Lists user-key assignments
RCLMENU	Recalls number and page of active menu
STOF	Sets system and user flags
STOKEYS	Makes multiple user-key assignments
TMENU	Displays temporary built-in or list-defined menu

DATA ENTRY AND EDITING	
CHOOSE	Displays a choose box with specified elements
FREEZE	Freezes up to three display areas
INFORM	Displays input form with specified fields
INPUT	Suspends program and waits for data
KEY	Returns key in buffer
LAST	Returns LAST arguments (if saved)
LASTARG	Returns LAST arguments (if saved)
MSGBOX	Displays message, waits for acknowledgment
NOVAL	No-Value placeholder for input form field data
PROMPT	Displays prompt and halts program
TEXT	Selects the stack display
WAIT	Pauses program execution or waits for a key

DEBUGGING AND ERROR HANDLING	
DOERR	Generates system or user-defined error
ERR0	Clears the last error number
ERRM	Returns the last error message
ERRN	Returns the last error number
HALT	Suspends program execution
IFERR	Begins IFERR test
KILL	Cancels all suspended programs
LAST	Returns arguments (if saved)
LASTARG	Returns arguments (if saved)

DIFFERENTIAL EQUATIONS	
RKF	Computes solution of initial value problem using RKF method
RKFERR	Computes the change in solution and absolute error estimate using the RKF method
RKFSTEP	Computes the next solution step of the initial value problem within a specified error tolerance using the RKF method
RRK	Computes solution of initial value problem using RRK method
RRKSTEP	Computes the next solution step of the initial value problem within a specified error tolerance using the RRD method
RSBERR	Computes the change in solution and absolute error estimate using the Rosenbrock and RKF methods

DISPLAY MANAGEMENT	
CLLCD	Clears the stack display
DISP	Displays an object on line n
FREEZE	Freezes up to three display areas
PICTURE	Enters the graphics environment
PVIEW	Displays <i>PICT</i> at specified coordinate
TEXT	Selects the stack display

EQUATION LIBRARY	
EQNLIB	Displays the Equation Library catalog
SOLVEQN	Places an equation set into the Solver

FINANCE	
AMORT	Computes amortization
TVM	Displays TVM solver menu
TVMBEG	Sets payment mode to beginning of period
TVMEND	Sets payment mode to end of period
TVMROOT	Solves for TVM variable

GENERAL MATH	
ABS	Absolute value
ARG	Returns the polar angle θ of a coordinate pair (x,y)
CEIL	Next greater integer
CONJ	Complex conjugate
FACT	Factorial or gamma function
FFT	Computes discrete Fourier transform
FLOOR	Next smaller integer
FP	Fractional part
HMS+	Adds in H.MS format
HMS-	Subtracts in H.MS format
HMS→	Converts a number from H.MS format
→HMS	Converts a number to H.MS format

GENERAL MATH (cont.)	
IFFT	Computes inverse discrete Fourier transform
INV	Inverse (reciprocal)
IP	Integer part
MANT	Mantissa of a number
MAX	Maximum of two numbers
MIN	Minimum of two numbers
MOD	Modulo
NEG	Negates an argument
PCOEF	Computes coefficients of a polynomial with specified roots
PEVAL	Evaluates a polynomial with specified coefficients at x
PROOT	Computes roots of polynomial with specified real or complex coefficients
→Q	Converts number to fractional equivalent
→Q π	→Q after factoring out π
RE	Real part of a complex number
RND	Rounds fractional part of number
ROOT	Finds a numerical root
RSD	Computes a correction to the solution of a system of equations
R→D	Radians-to-degrees conversion
SIGN	Sign of a number
SQ	Squares a number
TAYLR	Computes a Taylor series approximation
TRNC	Truncates number
XPON	Exponent of a number
XROOT	x^{th} root of y
$\sqrt{\quad}$	Square root
\int	Integral
∂	Derivative
+	Adds two objects
-	Subtracts two objects
*	Multiplies two objects
/	Divides two objects
^	Raises a number to a power
%	Percent
%CH	Percent change
%T	Percent total

GRAPHICS AND PLOTTING	
ANIMATE	Animates a series of grobs in <i>PICT</i>
ARC	Draws an arc in <i>PICT</i>
ATICK	Specifies axes tick mark spacing
AUTO	Scales y-axis
AXES	Sets intersection of axes and optionally stores labels
BAR	Selects bar plot
BARPLOT	Draws a bar plot of the data in ΣDAT

GRAPHICS AND PLOTTING (cont.)

BLANK	Creates a blank graphics object
BOX	Draws a box in <i>PICT</i>
CENTR	Sets center of plot display
CLLCD	Clears the stack display
CONIC	Selects conic plot
C→PX	User-unit to pixel coordinate conversion
DEPND	Specifies plot dependent column, variable, or range
DIFFEQ	Selects differential equation plot
DRAW	Draws a plot
DRAX	Draws axes
ERASE	Erases <i>PICT</i>
EYEPT	Specifies location of eyepoint relative to view volume
FUNCTION	Selects function plot
GOR	Superimposes graphics objects
GRAPH	Enters the graphics environment
GRIDMAP	Selects gridmap plot
→GROB	Converts object into graphics object
GXOR	Superimposes and inverts graphics objects
*H	Adjusts the height of a plot
HISTOGRAM	Selects histogram plot
HISTPLOT	Draws a histogram of the data in ΣDAT
INDEP	Selects plot independent column, variable or range
LABEL	Labels axes
LCD→	Returns LCD as 131x64 pixel graphics object
→LCD	Displays graphics object
LINE	Draws a line between two coordinates
NEG	Inverts a graphics object
NUMX	Specifies number of 3D plot steps in <i>x</i>
NUMY	Specifies number of 3D plot steps in <i>y</i>
PARAMETRIC	Selects parametric plot
PARSURFACE	Selects parsurface plot
PCONTOUR	Selects pseudo-contour plot
PDIM	Changes the size of <i>PICT</i>
PICT	Returns the name <i>PICT</i>
PICTURE	Enters the graphics environment
PIXOFF	Turns off a pixel in <i>PICT</i>
PIXON	Turns on a pixel in <i>PICT</i>
PIX?	Tests a pixel in <i>PICT</i>
PMAX	Specifies the upper-right plot coordinates
PMIN	Specifies the lower-left plot coordinates
POLAR	Selects polar plot
PRLCD	Prints an image of the display
PVIEW	Displays <i>PICT</i> at specified coordinate
PWRFIT	Selects power curve-fitting model
PX→C	Pixel to user-unit coordinate conversion

GRAPHICS AND PLOTTING (cont.)

RCEQ	Recalls the current equation
REPL	Writes one graphics object into another graphics object
RES	Sets the plot resolution in user unit or pixel intervals
SCALE	Specifies x and y scale in units per 10 pixels
SCATRLOT	Draws a scatter plot of the data in ΣDAT
SCATTER	Selects scatter plot
SIZE	Finds the dimensions of a graphics object
SLOPEFIELD	Selects slopefield plot
STEQ	Stores into reserved variable EQ
SUB	Extracts a sub-grob
TEXT	Displays the stack display
TLINE	Toggles pixels on a straight line
TRUTH	Selects truth plot
*W	Adjusts the width of a plot
WIREFRAME	Selects wireframe plot
XCOL	Specifies ΣDAT column as independent variable
XVOL	Specifies view volume width
XXRNG	Specifies width of 3D target mapping range
YCOL	Specifies a ΣDAT column as the dependent variable
YRNG	Specifies y-axis plotting range
YSLICE	Selects yslice plot
YVOL	Specifies view volume depth
YYRNG	Specifies height of 3D target mapping range

HYPERBOLIC OPERATIONS

ACOSH	Inverse hyperbolic cosine
ASINH	Inverse hyperbolic sine
ATANH	Inverse hyperbolic tangent
COSH	Hyperbolic cosine
EXPM	Natural exponential minus 1
LNP1	Natural logarithm of (argument + 1)
SINH	Hyperbolic sine
TANH	Hyperbolic tangent

INPUT/OUTPUT AND DATA TRANSFER

BAUD	Sets the baud rate
BEEP	Sounds a beep
BUFLN	Returns number of characters in the serial buffer
CHOOSE	Displays a choose box with specified elements
CKSM	Select the checksum scheme
CLOSEIO	Closes the serial port
FINISH	Terminates Kermit server mode
INFORM	Displays an input form with specified fields
INPUT	Suspends program and waits for data

INPUT/OUTPUT AND DATA TRANSFER (cont.)

KERRM	Returns the last Kermit error message
KEY	Returns key in buffer
KGET	Gets named data from a remote device
MSGBOX	Displays message, waits for acknowledgment
OPENIO	Opens IR or wired port
PARITY	Sets parity
PKT	Sends commands to server
RECN	Receives and renames file from remote Kermit
RECV	Receives file from remote Kermit, saves in a sender-named object
SBRK	Sends serial break
SEND	Sends object to another Kermit device
SERVER	Selects Kermit Server mode
SRECV	Reads characters from I/O port without Kermit
STIME	Sets serial transmit/receive timeout
TRANSIO	Selects character translation mode
XMIT	Sends string through I/O port without Kermit
XRECV	Receives an object using Xmodem protocol
XSEND	Sends an object using Xmodem protocol

LOGARITHMIC OPERATIONS

ALOG	Antilogarithm
e	Symbolic constant e
EXP	Natural exponential
EXPM	Natural exponential minus 1
LN	Natural logarithm
LNPI	Natural logarithm of (argument + 1)
LOG	Common (base 10) logarithm
XPON	Returns the exponent of a number

LOGICAL AND RELATIONAL OPERATORS

AND	Logical or binary AND
NOT	Logical or binary NOT
OR	Logical or binary OR
SAME	Tests two objects for equality
XOR	Logical or binary XOR
<	Less-than comparison
≤	Less-than-or-equal comparison
>	Greater-than comparison
≥	Greater-than-or-equal comparison
≠	Not-equal comparison
==	Tests two objects for equality

MATRIX AND ARRAY OPERATIONS

ABS	Square root of sum of squares of elements
ARRAY→	Separate array into individual elements
→ARRAY	Combines numbers into an array
C→R	Separates complex array into two arrays
CNRM	Column norm
COL+	Adds a column vector to a matrix
COL-	Deletes a column from a matrix
→COL	Separates a matrix into column vectors
COL→	Combines column vectors into a matrix
CON	Creates a constant array
COND	Estimates the column norm condition number of a matrix
CONJ	Complex conjugate
CROSS	Cross product
CSWP	Swaps two columns in a matrix
DET	Determinant of a matrix
→DIAG	Returns a vector of major diagonal elements
DIAG→	Creates a matrix with specified diagonal elements
DOT	Dot product of two vectors
EGV	Computes eigenvalues and right eigenvectors of a square matrix
EGVL	Computes eigenvalues of a square matrix
GET	Gets an element from a list or array
GETI	Gets an element from a matrix, increments and returns the index, and returns the matrix
IDN	Creates an identity matrix
IM	Returns array of imaginary parts from complex array
LQ	Returns the LQ factorization of a matrix
LSQ	Returns the minimum norm least-squares solution to a system of linear equations
LU	Returns the Crout LU decomposition of a square matrix
NEG	Negates elements in an array
PUT	Replaces an element in an list or array
PUTI	Replaces an element in an list or array, increments and returns the index, and returns the list or array
QR	Computes the QR factorization of a matrix
R→C	Combines two arrays into complex array
RANK	Estimates the rank of a rectangular matrix
RANM	Creates an array of random integers
RCI	Multiplies a row in a matrix by a factor
RCIJ	Multiplies a row in a matrix by a factor and adds the result to elements in another row
RDM	Redimensions an array
RE	Returns array of real parts from complex array
RNRM	Computes row norm of an array
ROW+	Adds a row to a matrix
ROW-	Deletes a row from a matrix
→ROW	Separates a matrix into row vectors
ROW→	Combines row vectors into a matrix

MATRIX AND ARRAY OPERATIONS (cont.)

RREF	Computes the reduced row-echelon form of a matrix
RSWP	Swaps two rows in a matrix
SCHUR	Computes the Schur decomposition of a square matrix
SIZE	Finds the number of elements in an array or matrix
SNRM	Computes the spectral norm of an array
SQ	Squares a matrix
SRAD	Computes the spectral radius of a square matrix
SVD	Computes the singular value decomposition of a matrix
SVL	Computes the singular values of a matrix
TRACE	Computes the trace of a square matrix
TRN	Transposes a matrix
→V2	Combines two real numbers into a vector
→V3	Combines three real numbers into a vector
V→	Separates a 2 or 3 element vector

MEMORY MANAGEMENT

ARCHIVE	Makes backup copy of HOME directory
ATTACH	Attaches library to current directory
BYTES	Returns the checksum and number of bytes of an object
CLTEACH	Purges the examples directory
CLUSR	Purges all user variables in the current directory
CLVAR	Purges all user variables in the current directory
CRDIR	Creates a directory
DEFINE	Creates user-defined function
DETACH	Detaches library from current directory
FREE	Frees merged memory
FREE1	Frees merged memory in port 1
HOME	Selects the HOME directory
LIBS	Lists libraries attached to current directory
MEM	Returns available memory
MERGE	Merges RAM card with main memory
MERGE1	Merges RAM card in port 1 with main memory
NEWOB	Separates object from list or backup name
ORDER	Rearranges the VAR menu
PATH	Returns a list showing the current path
PGDIR	Purges specified directory and its contents
PINIT	Initializes a RAM card in port 2
PURGE	Purges one or more variables
PVARs	Returns list of port objects
RCEQ	Recalls the current equation
RCL	Recalls the contents of a variable
RCLF	Returns a list containing the system and user flags
RCLΣ	Recalls the current statistics matrix
RESTORE	Replaces HOME directory with backup copy

MEMORY MANAGEMENT (cont.)

SAME	Tests two objects for equality
SIZE	Finds the dimensions of an object
STEQ	Stores into reserved variable <i>EQ</i>
STO	Stores an object into a variable
STO Σ	Stores into reserved variable Σ <i>DAT</i>
TEACH	Creates examples directory in the VAR menu
TVAR\$	Lists the variables of specified type
TYPE	Returns the type of an object
UPDIR	Makes parent directory the current directory
VARS	Returns list of variables in the current directory
VTYPE	Returns type of object in named variable
→	Assigns local variable(s)

MISCELLANEOUS FUNCTIONS & COMMANDS

DARCY	Calculates Darcy friction factor
FO λ	Calculates black-body emissive power
MINEHUNT	Starts the Minehunt game
SIDENS	Calculates intrinsic density of silicon
VERSION	Displays operating system version and copyright
ZFACTOR	Calculates gas compressibility factor Z

MODES AND FLAGS

BIN	Sets binary base
CF	Clears a system or user flag
CYLIN	Sets polar/cylindrical angle mode
DEC	Sets decimal base
DEG	Sets Degrees mode
ENG	Sets Engineering display mode
FC?	Tests a system or user flag
FC?C	Tests and clears a system or user flag
FIX	Sets Fix display mode
FS?	Tests a system or user flag
FS?C	Tests and clears a system or user flag
GRAD	Sets Grads mode
HEX	Sets hexadecimal base
OCT	Sets octal base
RAD	Sets Radians mode
RCLF	Returns a list containing the system and user flags
RECT	Sets rectangular angle mode
SCI	Sets Scientific display mode
SF	Sets a system or user flag
SPHERE	Sets polar/spherical angle mode
STD	Sets Standard display mode
STOF	Sets system and user flags

MULTIPLE EQUATION SOLVER

MCALC	Sets Multiple Equation Solver variable to <i>not</i> user-defined state
MINIT	Establishes <i>Mpar</i> from <i>EQ</i>
MITM	Specifies the title and order of menu in <i>Mpar</i>
MROOT	Solves for single or all variables using the Multiple Equation Solver
MSOLVR	Displays the Multiple Equation Solver menu
MUSER	Sets Multiple Equation Solver variable to user-defined state

PRINTING

CR	Prints a carriage-right
DELAY	Sets $0 \leq n \leq 6.9$ sec delay between printed lines
OLDPRT	Remaps to HP 82240A character set
PRLCD	Prints an image of the display
PRST	Prints the stack
PRSTC	Prints the stack in compact format
PRVAR	Prints the name and contents of one or more variables
PR1	Prints an object

PROBABILITY

COMB	Combinations of n objects taken r at a time
FACT	Factorial or gamma function
!	Factorial or gamma function
PERM	Permutations of n objects taken r at a time
RAND	Returns a random number
RDZ	Sets the random number seed
UTPC	Upper-tail Chi-Square distribution
UTPF	Upper-tail F-distribution
UTPN	Upper-tail normal distribution
UTPT	Upper-tail t-distribution

PROGRAM BRANCHING AND CONTROL

CASE	Begins CASE structure
CONT	Continues a halted program
DO	Begins DO loop
DOERR	Generates user-defined or system error
ELSE	Begins ELSE clause
END	Ends program structures
EVAL	Evaluates an object
FOR	Begins FOR loop
HALT	Suspends program execution
IF	Begins IF test
IFERR	Begins IFERR test
IFT	IF ... THEN ... END test
IFTE	IF ... THEN ... ELSE ... END test

PROGRAM BRANCHING AND CONTROL (cont.)

INPUT	Suspends program and waits for data
KILL	Cancels all suspended programs
LIBEVAL	Executes a library object
NEXT	Ends FOR ... NEXT or START ... NEXT
→NUM	Evaluates an object to yield a numeric result
OFF	Turns the calculator off
PROMPT	Displays prompt and halts program
REPEAT	Part of WHILE ... REPEAT ... END
START	Begins START ... NEXT or START ... STEP
STEP	Ends FOR ... STEP or START ... STEP
SYSEVAL	Executes a system object
THEN	Begins THEN clause
UNTIL	Part of DO ... UNTIL ... END
UPDIR	Makes parent directory current directory
WAIT	Pauses program execution or waits for a key
WHILE	Begins WHILE ... REPEAT ... END
WSLOG	Returns the four most recent system halts
→	Assigns local variable(s)

STACK MANIPULATION

→ARRAY	Combines numbers into an array
CLEAR	Clears the stack
DEPTH	Counts the objects on the stack
DROP	Drops one object from the stack
DROPN	Drops $n+1$ objects from the stack
DROP2	Drops two objects from the stack
DUP	Duplicates one object on the stack
DUPN	Duplicates n objects on the stack
DUP2	Duplicates two objects on the stack
LAST	Returns LAST arguments (if saved)
LASTARG	Returns LAST arguments (if saved)
→LIST	Combines objects into a list
OVER	Copies the object in level 2 into level 1
PICK	Copies n th object into level 1 (excluding n)
ROLL	Moves level $n+1$ object to level 1 (excluding n)
ROLLD	Moves the level 2 object to level n (excluding n)
ROT	Moves the level 3 object to level 1
SWAP	Swaps the objects in levels 1 and 2

STATISTICS	
BAR	Selects bar plot
BARPLOT	Draws a bar plot of the data in ΣDAT
BESTFIT	Executes LR and computes the best curve fit
BINS	Sorts ΣDAT data into histogram bins
CL Σ	Purges the statistics matrix
CNRM	Column norm of an array
COL Σ	Specifies dependent and independent columns in ΣDAT
CORR	Correlation coefficient
COV	Sample covariance
EXPFIT	Selects exponential curve-fitting model
HISTOGRAM	Selects histogram plot
HISTPLOT	Draws a histogram of the data in ΣDAT
LINFIT	Selects linear curve-fitting model
LOGFIT	Selects logarithmic curve-fitting model
LR	Linear regression
MAX Σ	Finds the maximum coordinate values in ΣDAT
MEAN	Means of the data in ΣDAT
MIN Σ	Finds the minimum coordinate values in ΣDAT
NDIST	Normal probability density
N Σ	Number of data points in ΣDAT
PCOV	Population covariance
PVAR	Population variances of the data in ΣDAT
PREDV	Predicted dependent variable value
PREDX	Predicted independent variable value
PREDY	Predicted dependent variable value
PSDEV	Population standard deviation of the data in ΣDAT
PWRFIT	Selects power curve-fitting model
RCL Σ	Recalls the current statistics matrix
SCATRPLOT	Draws a scatter plot of the data in ΣDAT
SCATTER	Selects scatter plot
SDEV	Sample standard deviations of the data in ΣDAT
STO Σ	Stores into reserved variable ΣDAT
TOT	Sums the columns in ΣDAT
VAR	Sample variances of the data in ΣDAT
XCOL	Specifies a ΣDAT column as the independent variable
YCOL	Specifies a ΣDAT column as the dependent variable
XRNG	Specifies x-axis plotting range
Σ	Summation
$\Sigma LINE$	Best-fit line for data in ΣDAT
ΣX	Sum of data in independent ΣDAT column
ΣX^2	Sum of squares in independent ΣDAT column
ΣY	Sum of data in dependent ΣDAT column
ΣY^2	Sum of squares of data in dependent ΣDAT column
$\Sigma X*Y$	Sum of products in independent and dependent ΣDAT columns
$\Sigma +$	Appends one or more data points to ΣDAT
$\Sigma -$	Deletes last row from ΣDAT

STRING MANIPULATION

CHR	Makes a one-character string
HEAD	Returns the first character of a string
NUM	Returns character code of a string's first character
POS	Finds a substring in a string
SIZE	Finds the number of characters in a string
STR→	Parses and evaluates a string
→STR	Converts an object to a string
SUB	Extracts a portion of a string
TAIL	Returns a string less its first character

SYMBOLIC MANIPULATION

APPLY	Returns an evaluated expression as the argument to an unevaluated local name
COLCT	Collects like terms
EQ→	Separates equation into left and right sides
EXPAN	Expands an algebraic
e	Symbolic constant e
i	Symbolic constant i
π	Symbolic constant π
ISOL	Isolates a variable in an equation
LININ	Test if an equation is linear in a variable
↑MATCH	Match-and-replace, beginning with subexpressions
↓MATCH	Match-and-replace, beginning with the top-level expression
→NUM	Evaluates an object to yield a numeric result
OBJ→	Separates outermost function and its arguments
QUAD	Solves a quadratic polynomial
QUOTE	Returns argument expression unevaluated
SHOW	Resolves all references to a name implicit in an algebraic
TAYLR	Computes a Taylor series approximation
\int	Integral
∂	Derivative
	"Where": appends local name and value to evaluated expression

TRIGONOMETRIC OPERATIONS

ACOS	Arc cosine
ASIN	Arc sine
ATAN	Arc tangent
COS	Cosine
D→R	Degrees-to-radians conversion
R→D	Radians-to-degrees conversion
SIN	Sine
TAN	Tangent

TIME AND ALARMS

ACK	Acknowledges displayed past due alarm
ACKALL	Acknowledges all past due alarms
CLKADJ	Add clock ticks to the system time
DATE	Returns the system date
→DATE	Sets the system date
DATE+	Adds a number of days to a date
DDAYS	Number of days between two dates
DELALARM	Deletes an alarm
FINDALARM	Returns alarm index <i>n</i>
HMS+	Adds in H.MS format
HMS-	Subtracts in H.MS format
HMS→	Converts a number from H.MS format
→HMS	Converts a number to H.MS format
RCLALARM	Recalls alarm from alarm list
STOALARM	Stores alarm in system alarm list
TICKS	Returns time in binary integer clock ticks
TIME	Returns current time as number
→TIME	Sets specified system time
TSTR	Converts date & time numbers to string form

UNIT OBJECT OPERATIONS

CONVERT	Performs a unit conversion
OBJ→	Decomposes a unit object into a number and unit expression
TDELTA	Calculates temperature difference
TINC	Adds temperature increment
UBASE	Converts unit object to SI base units
UFACTOR	Factors specified compound unit
→UNIT	Builds a unit object
UVAL	Returns scalar portion of unit object

VARIABLE ARITHMETIC

DECR	Decrements then returns value of a variable
INCR	Increments then returns value of a variable
SCONJ	Conjugates the contents of a variable
SINV	Inverts the contents of a variable
SNeg	Negates the contents of a variable
STO+	Storage arithmetic add
STO-	Storage arithmetic subtract
STO*	Storage arithmetic multiply
STO/	Storage arithmetic divide

Command Reference

This command reference provides information about all commands and functions in the HP 48. Each entry lists the name, menu locations, characteristics, description, stack diagrams, and related flags if applicable.

NAME	KEY	MENU	Characteristics
Description			
		Input	Output
		Level ₃ Level ₂ Level ₁	→ Level ₃ Level ₂ Level ₁
<i>Related Flags:</i> Flags which may affect the result			
<i>Notes:</i> Notes about the command or function			

The characteristics are encoded as follows:

Symbol	Characteristic
G	New HP 48G/GX command
{ }	Accepts arguments in lists
↓	Invertible
∂	Differentiable
∫	Integrable

For instance, ACOSH is a function which has an inverse, is differentiable, and can accept arguments in a list:

ACOSH	MTH	NXT	HYP	ACOSH	{ } ↓ ∂	Function
Inverse hyperbolic cosine						
			z	→	acosh z	
			'symb'	→	'ACOSH(symb)'	
Related Flags: -1, -3						

Commands or functions which can take their arguments in lists (indicated by a { } characteristic) execute once for each argument in the list. For instance, the program « { 3 5 } SF » sets user flags 3 and 5. The program « { 1 2 6 } { 3 7 8 } * » returns the list { 3 14 48 }.

The following table lists the terms used in the stack diagrams. Note that system modes may affect the interpretation of input parameters or the results of some functions.

Term	Description
obj	Any object
x or y	Real number
a b c d	Real number
(x,y)	Complex number or user-unit graphics coordinates
z	Real or complex number
m or n	Positive integer real number (rounded if non-integer)
#n or #m	Binary integer
x_unit	Real number with units
"string"	Character string
{ list }	List of objects
grob	Graphics object
{ #x #y }	Pixel coordinates
hms	Real number in HH.MMSS format
time	Time in HH.MMSS format
date	Date in current MM.DDYYYY or DD.MMYYYY format (flag -42)
T/F	Test result: 0 (false) or non-zero (true)
'symb'	Expression or name treated as an algebraic
[vector]	Real or complex vector
[[matrix]]	Real or complex matrix
[R-array]	Real vector or matrix
[C-array]	Complex vector or matrix
{row col}	Coordinates of an element in a matrix
position	Real number specifying an element in a list, vector, or matrix. May be a list containing two real numbers specifying an element in a matrix.
'name'	Global or local name
'global'	Global name
rc or rc.p	Key location: row-col or row-col.plane (see <i>User Keys</i>)
mm.pp	Menu specified as menu.page
d.o.f.	Degrees of freedom (positive integer)
port	Port number: 0 – 33 or & (wildcard)
backup	Backup object
library	Library object
LID	Library identifier (port:library number)

ABS	[MTH] VECTR ABS { } ∂ Function
	[MTH] MATR NORM ABS
	[MTH] REAL [NXT] ABS
	[MTH] [NXT] CMPL ABS
Absolute value. For arrays, returns the norm, defined as the square root of the sum of squares of the absolute values of the elements.	
	$x \rightarrow x $
	$(x,y) \rightarrow \sqrt{x^2 + y^2}$
	[vector] $\rightarrow \text{vector} $
	[[matrix]] $\rightarrow \text{matrix} $
	'symb' $\rightarrow \text{'ABS(symb)'}$
	x_unit $\rightarrow x _{\text{unit}}$
<i>Related Flag: -3</i>	
ACK	[↵] [TIME] ALRM ACK Command
Acknowledges displayed past due alarm	
<i>Related Flags: -43, -44</i>	
ACKALL	[↵] [TIME] ALRM ACKA Command
Acknowledges all past due alarms	
<i>Related Flags: -43, -44</i>	
ACOS	[↵] [ACOS] { } ↓ ∂ Function
Arc cosine	
	$z \rightarrow \text{acos } z$
	'symb' $\rightarrow \text{'ACOS(symb)'}$
<i>Related Flags: -1, -3, -17, -18</i>	
ACOSH	[MTH] HYP ACOSH { } ↓ ∂ Function
Inverse hyperbolic cosine	
	$z \rightarrow \text{acosh } z$
	'symb' $\rightarrow \text{'ACOSH(symb)'}$
<i>Related Flags: -1, -3</i>	

ADD**[MTH]** LIST ADD

G ↓ ∂ Function




Adds two objects

z_1	z_2	→	z_1+z_2
#n	m	→	#n+m
n	#m	→	#n+m
#n	#m	→	#n+m
x_unit	y_unit	→	x+y_unit
'symb ₁ '	'symb ₂ '	→	'symb ₁ +symb ₂ '
z	'symb'	→	'z+symb'
'symb'	z	→	'symb+z'
'symb'	x_unit	→	'symb+x_unit'
x_unit	'symb'	→	'x_unit+symb'
[vector ₁]	[vector ₂]	→	[vector ₁ +vector ₂]
[[matrix ₁]]	[[matrix ₂]]	→	[[matrix ₁ +matrix ₂]]
grob ₁	grob ₂	→	grob ₃
"abc"	"def"	→	"abcdef"
"string"	object	→	"stringobject"
object	"string"	→	"objectstring"
{ obj ₁ obj ₂ }	{ obj ₃ obj ₄ }	→	{ obj ₁ +obj ₃ obj ₂ +obj ₄ }
{ obj ₁ obj ₂ ... }	obj _{new}	→	{ obj ₁ +obj _{new} obj ₂ +obj _{new} ... }
obj _{new}	{ obj ₁ obj ₂ ... }	→	{ obj _{new} +obj ₁ obj _{new} +obj ₂ ... }

Related Flags: -3, -5 through -10*Notes:*

- 1) ADD is the same as +, except that element-wise addition is performed on lists instead of concatenating lists. The lists must contain the same number of objects.
- 1) Grobs must have the same dimensions
- 2) →STR is executed on objects added to strings
- 3) Units must be dimensionally consistent

ALOG		{ } ↓ ∂ } Function
Antilogarithm		
$z \rightarrow 10^z$ $\text{'symb'} \rightarrow \text{'ALOG(symb)'}$		
<i>Related Flag:</i> -3		
AMORT	TVM AMOR	G Command
Calculates amortization from TVM variables I%YR, PV, PMT, FV, and PYR.		
payments → principal interest balance		
<i>Related Flag:</i> -14		
AND	BASE LOGIC AND { } Function	
Logical or binary AND		
$\#n_1 \#n_2 \rightarrow \#n_3$ $x \ y \rightarrow \text{T/F}$ $x \ \text{'symb'} \rightarrow \text{'x AND symb'}$ $\text{'symb'} \ x \rightarrow \text{'symb AND x'}$ $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'symb}_1 \text{ AND } \text{'symb}_2$ $\text{"string}_1 \ \text{"string}_2 \rightarrow \text{"string}_3$		
<i>Related Flags:</i> -3, -5 through -10		
<i>Note:</i> String arguments must have the same length		
ANIMATE	GROB ANIM	G Command
Animates a series of grobs. The grobs may optionally be specified as names. Delay is the delay in seconds between each grob displayed. Repeat is the number of times to repeat the sequence (1048575 cycles if repeat=0).		
$\text{grob}_1 \dots \text{grob}_n \ n \rightarrow \text{grob}_1 \dots \text{grob}_n \ n$ $\text{name}_1 \dots \text{name}_n \ n \rightarrow \text{grob}_1 \dots \text{grob}_n \ n$ $\text{grob}_1 \dots \text{grob}_n \ \{ \ n \ \{ \ \#x \ \#y \ \} \ \text{delay} \ \text{repeat} \ \} \rightarrow \text{grob}_1 \dots \text{grob}_n \ \{ \}$ $\text{name}_1 \dots \text{name}_n \ \{ \ n \ \{ \ \#x \ \#y \ \} \ \text{delay} \ \text{repeat} \ \} \rightarrow \text{grob}_1 \dots \text{grob}_n \ \{ \}$		
<i>Notes:</i>		
1) If the list specifier is not used, the delay between each grob will default to .1 second.		
2) ANIMATE leaves its parameters on the stack.		










APPLY	 SYMBOLIC NXT APPLY	∂ Function
Returns an evaluated expression as the argument to an unevaluated local name $\{ \text{symb}_1 \dots \text{symb}_n \} \text{'name'} \rightarrow \text{'name}(\text{symb}_1, \dots, \text{symb}_n)'$ $\text{'APPLY}(\text{symb}_1, \dots, \text{symb}_n)'$		
ARC	PRG PICT ARC	Command
Draws an arc in <i>PICT</i> centered at (x,y), radius <i>r</i> , counterclockwise from θ_1 to θ_2 $(x,y) \ r \ \theta_1 \ \theta_2 \rightarrow$ $\{ \#x \ \#y \} \ \#r \ \theta_1 \ \theta_2 \rightarrow$ <i>Related Flags:</i> -17, -18		
ARCHIVE	 MEMORY NXT ARCHI	$\{ \}$ Command
Makes a backup copy of HOME directory $\text{:IO: name} \rightarrow$ $\text{:port: name} \rightarrow$ <i>Related Flags:</i> -33, -39 <i>Note:</i> ARCHIVE does not save the settings of user flags or the contents of port 0.		
ARG	MTH NXT CMPL ARG	$\{ \}$ ∂ Function
Returns the polar angle θ of a coordinate pair (x,y) $z \rightarrow \theta$ $\text{'symb'} \rightarrow \text{'ARG}(\text{symb})'$ <i>Related Flags:</i> -17, -18		
ARRY→		$\{ \}$ Command
Separates array into individual elements $[\text{vector}] \rightarrow z_1 \dots z_n \ \{n\}$ $[[\text{matrix}]] \rightarrow z_{11} \ z_{12} \dots z_{nm} \ \{n \ m\}$		
→ARRY	PRG TYPE →ARR	Command
Combines real or complex numbers into an array $z_1 \dots z_n \ \{n\} \rightarrow [\text{vector}]$ $z_{11} \ z_{12} \dots z_{nm} \ \{n \ m\} \rightarrow [[\text{matrix}]]$		
ASIN	 ASIN	$\{ \} \downarrow \partial$ Function
Arc sine $z \rightarrow \text{asin } z$ $\text{'symb'} \rightarrow \text{'ASIN}(\text{symb})'$ <i>Related Flags:</i> -1, -3, -17, -18		

ASINH	(MTH)	HYP	ASINH	{ } ↓ ∂	Function
Inverse hyperbolic sine					
$z \rightarrow \sinh z$					
'symb' \rightarrow 'ASINH(symb)'					
<i>Related Flags:</i> -1, -3					
ASN	(↵)	(MODES)	KEYS	ASN	{ } Command
Make a single user-key assignment					
object rc.p \rightarrow					
'SKEY' rc.p \rightarrow <i>Reactivates standard key</i>					
<i>Related Flags:</i> -61, -62					
ASR	(MTH)	BASE	(NXT)	BIT	ASR { } Command
Arithmetic shift right (preserves most significant bit)					
$\#n_1 \rightarrow \#n_2$					
<i>Related Flags:</i> -5 through -12					
ATAN	(↵)	(ATAN)		{ } ↓ ∂	Function
Arc tangent					
$z \rightarrow \operatorname{atan} z$					
'symb' \rightarrow 'ATAN(symb)'					
<i>Related Flags:</i> -1, -3, -17, -18					
ATANH	(MTH)	HYP	ATANH	{ } ↓ ∂	Function
Inverse hyperbolic tangent					
$z \rightarrow \operatorname{atanh} z$					
'symb' \rightarrow 'ATANH(symb)'					
<i>Related Flags:</i> -1, -3, -22					
ATICK	(↵)	(PLOT)	PPAR	(NXT)	ATICK G Command
Specifies tick spacing on plot axes					
$n \rightarrow$ <i>User-unit spacing for both axes</i>					
{ x y } \rightarrow <i>User-unit spacing for each axis</i>					
$\#n \rightarrow$ <i>Pixel spacing for both axes</i>					
{ #x #y } \rightarrow <i>Pixel spacing for each axis</i>					
ATTACH	(↵)	(LIBRARY)	(NXT)	ATTAC	{ } Command
Attaches library to current directory					
LID \rightarrow					
<i>Note:</i> Many libraries may be attached to the HOME directory, but only one library at a time may be attached to a subdirectory.					

AUTO	[PLOT] [NXT] AUTO	Command
Scales y-axis		
AXES	[PLOT] PPAR [NXT] AXES	Command
Sets intersection of axes and optionally stores labels and tick spacing		
	(x,y) →	<i>Replaces entire axes parameter</i>
	{ (x,y) } →	<i>Replaces intersection only</i>
	{ "Xlabel" "Ylabel" } →	<i>Specifies labels only</i>
	{ (x,y) "Xlabel" "Ylabel" } →	<i>Replaces entire axes parameter</i>
	{ tick } →	<i>Specifies tick spacing only</i>
	{ (x,y) tick } →	<i>Replaces entire axes parameter</i>
	{ (x,y) tick "Xlabel" "Ylabel" } →	<i>Replaces entire axes parameter</i>
<i>Note: "tick" specifies the spacing of tick marks on the axes, and can have the following forms:</i>		
	x	<i>User-unit spacing for both axes</i>
	#n	<i>Pixel spacing for both axes</i>
	{ x y }	<i>User-unit spacing for x- and y-axes</i>
	{ #x #y }	<i>Pixel spacing for x- and y-axes</i>
BAR	[PLOT] [NXT] STAT PTYPE BAR	Command
Selects bar plot		
BARPLOT	[PLOT] PLOT BARPL	Command
Draws a bar plot of the data in ΣDAT		
BAUD	[I/O] IOPAR BAUD	Command
Sets the serial baud rate: 1200, 2400, 4800, or 9600 (default)		
	n →	
<i>Note: The clock should not be displayed during 9600 baud transfers</i>		
BEEP	[PRG] [NXT] OUT [NXT] BEEP	{ } Command
Sounds a beep. Maximum 4400 Hz, 1048 seconds.		
	Hz secs →	
<i>Related Flag: -56</i>		
BESTFIT	[STAT] SPAR MODL BEST	Command
Selects the statistics model that yields the largest absolute value for the correlation coefficient and executes the LR command		
BIN	[MTH] BASE [NXT] BIN	Command
Sets binary base		
<i>Related Flags: -5 through -12</i>		

BINS	[STAT] <code>IVAR BINS</code>	Command
Sorts the Σ <i>DAT</i> data into N bins using the independent variable column as the sort key. The level 1 result shows the number of data points less than and greater than the available bins.		
X_{\min} width N \rightarrow $[[b_1] \dots [b_N]]$ $[b_L b_R]$		
BLANK	[PRG] <code>GROB BLAN</code>	{ } Command
Creates a blank graphics object		
#width #height \rightarrow grob		
BOX	[PRG] <code>PICT BOX</code>	Command
Draws a box in <i>PICT</i> with opposite corners defined by user-unit or pixel coordinates		
(x,y) (x',y') \rightarrow		
{ #x #y } { #x' #y' } \rightarrow		
BUFLEN	[I/O] [NXT] <code>SERIA BUFLE</code>	Command
Returns the number of characters in the serial buffer and 1 if no error occurred.		
\rightarrow n T/F		
BYTES	[MEMORY] <code>BYTES</code>	Command
Returns the checksum and number of bytes of an object		
'global' \rightarrow #checksum size <i>object + name</i>		
object \rightarrow #checksum size <i>object only</i>		
B→R	[MTH] <code>BASE B→R</code>	{ } Command
Binary to real conversion		
#n \rightarrow n		
<i>Related Flags:</i> -5 through -12		
CASE	[PRG] <code>BRCH CASE CASE</code>	Command
Begins CASE structure		
CASE		
test ₁ THEN action ₁ END		
test ₂ THEN action ₂ END		
...		
test _n THEN action _n END		
default-action (optional)		
END		





CEIL	[MTH]	REAL	[NXT]	[NXT]	CEIL	{ }	Function
Next greatest integer							
				x	\rightarrow	n	
				'symb'	\rightarrow	'CEIL(symb)'	
				x_unit	\rightarrow	n_unit	
<i>Related Flag: -3</i>							
CENTR	[↵]	[PLOT]	PPAR	[NXT]	CENT		Command
Sets center of plot display. Supplying x implies (x,0).							
				(x,y)	\rightarrow		
				x	\rightarrow		
CF	[PRG]	TEST	[NXT]	[NXT]	CF	{ }	Command
	[↵]	[MODES]	FLAG	CF			
Clears a system or user flag							
				$\pm n$	\rightarrow		
CHOOSE	[PRG]	[NXT]	IN	CHOOS			G Command
Displays a choose box with the highlight positioned at the specified element. For further details, see <i>Data Entry</i> .							
				"title" { list }	start \rightarrow	obj 1 OK	
				"title" { {disp return} ... }	start \rightarrow	0 CANCEL or CANCEL	
CHR	[↵]	[CHARS]	CHR			{ }	Command
	[PRG]	TYPE	[NXT]	CHR			
Makes a one-character string							
				n	\rightarrow	"string"	
CKSM	[↵]	[I/O]	IOPAR	CKSM			Command
Selects the checksum scheme							
				n	\rightarrow		
				1		1-digit arithmetic	
				2		2-digit arithmetic	
				3		3-digit CRC (default)	
CLEAR	[↵]	[CLEAR]					Command
Clears the stack							
				objects	\rightarrow		
CLKADJ	[↵]	[TIME]	[NXT]	[NXT]	CLKA	{ }	Command
Adds (subtracts) clock ticks to (from) the system time (8192 ticks/sec)							
				$\pm ticks$	\rightarrow		
CLLCD	[PRG]	[NXT]	OUT	CLLCD			Command
Clears the stack display							

CLOSEIO  I/O  NXT CLOSE	Command
Closes the serial port, clears input buffer and KERRM	
CLTEACH	Command
Purges the EXAMPLES directory	
CLUSR CLVAR	Command
Purges all user variables in the current directory	
CLΣ  STAT DATA CLΣ	Command
Purges the statistics matrix ΣDAT	
CNRM  MTH MATR NORM CNRM { }	Command
Computes the maximum value of the sums of the absolute values of all elements over all columns <div style="text-align: right;">[vector] → column-norm [[matrix]] → column-norm</div> <i>Note:</i> Since a vector is considered a 1-row matrix, CNRM returns the sum of the absolute values of the elements in the vector.	
COL+  MTH MATR COL COL+ G { }	Command
Inserts a column vector into a matrix or a number into a vector <div style="text-align: right;">[[matrix]] [vector] index → [[matrix]]' [vector] z index → [vector]'</div>	
COL-  MTH MATR COL COL- G { }	Command
Deletes a column from a matrix or a number from a vector <div style="text-align: right;">[[matrix]] index → [[matrix]]' [deleted_column] [vector] index → [vector]' deleted_element</div>	
COL→  MTH MATR COL COL→ G	Command
Transforms a series of column vectors into a matrix <div style="text-align: right;">[vector₁] ... [vector_n] n → [[matrix]]</div>	
→COL  MTH MATR COL →COL G { }	Command
Transforms a matrix into a series of column vectors <div style="text-align: right;">[[matrix]] → [vector₁] ... [vector_n] n</div>	
COLCT  SYMBOLIC COLCT { }	Command
Collects like terms <div style="text-align: right;">z → z 'symb₁' → 'symb₂'</div>	

COLΣ		Command
Specifies dependent and independent columns in ΣDAT independent dependent \rightarrow		
COMB	(MTH) (NXT) PROB COMB	{ } Function
Combinations of n objects taken m at a time $n \ m \rightarrow C_{n,m}$ $\text{'symb'} \ n \rightarrow \text{'COMB(symb,n)'}$ $n \ \text{'symb'} \rightarrow \text{'COMB(n,symb)'}$ $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'COMB(symb}_1, \text{symb}_2)'$		
<i>Related Flag: -3</i>		
CON	(MTH) MATR MAKE CON	Command
Creates a constant array or replaces the contents of an existing array or named array $\{ \text{rows cols} \} \ z \rightarrow [[\text{matrix}]]$ $[\text{vector}_1] \ z \rightarrow [\text{vector}_2]$ $[[\text{matrix}_1]] \ z \rightarrow [[\text{matrix}_2]]$ $\text{'name'} \ z \rightarrow$		
COND	(MTH) MATR NORM COND	G { } Function
Estimates the column norm condition number of a square matrix $[[\text{matrix}]] \rightarrow \text{condition-number}$		
CONIC	(G) (PLOT) P TYPE CONIC	Command
Selects conic plot		
CONJ	(MTH) (NXT) CMPL (NXT) CONJ	{ } $\downarrow \partial$ Function
Complex conjugate $x \rightarrow x$ $(x,y) \rightarrow (x,-y)$ $[\text{R-array}] \rightarrow [\text{R-array}]$ $[\text{C-array}_1] \rightarrow [\text{C-array}_2]$ $\text{'symb'} \rightarrow \text{'CONJ(symb)'}$		
<i>Related Flag: -3</i>		
CONLIB	(G) (EQ LIB) COLIB CONLI	G Command
Displays the Constants Library catalog		
CONST	(G) (EQ LIB) COLIB CONS	G { } Function
Returns the value of the specified constant $\text{'constname'} \rightarrow \text{constant}$		
<i>Related Flags: 60, 61</i>		

CONT	CONT	Command
Continues a halted program		
CONVERT	UNITS CONV	{ } Command
Performs a unit conversion		
$x_{old} \ y_{new} \rightarrow x'_{new}$ $x \ y \rightarrow x$		
CORR	STAT FIT CORR	Command
Correlation coefficient of ΣDAT data in columns specified by COL Σ		
\rightarrow correlation		
COS	COS	{ } $\downarrow \partial$ Function
Cosine		
$z \rightarrow \cos z$ $'symb' \rightarrow 'COS(symb)'$		
<i>Related Flags:</i> -3, -17, -18		
COSH	MTH HYP COSH	{ } $\downarrow \partial$ Function
Hyperbolic cosine		
$z \rightarrow \cosh z$ $'symb' \rightarrow 'COSH(symb)'$		
<i>Related Flag:</i> -3		
COV	STAT FIT COV	Command
Sample covariance of ΣDAT data in columns specified by COL Σ		
\rightarrow covariance		
CR	I/O PRINT CR	Command
Prints a carriage-right		
<i>Related Flags:</i> -33, -34, -37		
CRDIR	MEMORY DIR CRDIR	{ } Command
Creates a directory		
$'name' \rightarrow$		
CROSS	MTH VECTR CROSS	{ } Command
Cross product		
$[A] [B] \rightarrow [A \times B]$		
CSWP	MTH MATR COL CSWP	G { } Command
Column swap		
$[[matrix]] \ index_1 \ index_2 \rightarrow [[matrix]]'$		






CYLIN	[F] [MODES] ANGL RECT [MTH] VECTR [NXT] CYLIN	G Command
Sets polar/cylindrical coordinate mode		
C→PX	[PRG] PICT [NXT] C→PX	{ } Command
User-unit to pixel coordinate conversion (x,y) → { #x #y }		
<i>Note: Scaling information is derived from PPAR</i>		
C→R	[MTH] [NXT] CMPL C→R [PRG] TYPE [NXT] C→R	{ } Command
Complex-to-real conversion (x,y) → x y [C-array] → [R-array _{real}] [R-array _{imag}]		
DARCY	[F] [EQ LIB] UTILS DARCY	G { } Function
Calculates Darcy friction factor e/D Re → d 'symb' x → 'DARCY(symb,x)' x 'symb' → 'DARCY(x,symb)' 'symb ₁ ' 'symb ₂ ' → 'DARCY(symb ₁ ,symb ₂)'		
<i>Related Flag: -3</i>		
DATE	[F] [TIME] DATE	Command
Returns the system date → date		
<i>Related Flag: -42</i>		
→DATE	[F] [TIME] →DAT	Command
Sets the system date date →		
<i>Related Flag: -42</i>		
DATE+	[F] [TIME] [NXT] DATE+	{ } Command
Adds a number of days to a date date ±days → date'		
<i>Related Flag: -42</i>		
DDAYS	[F] [TIME] [NXT] DDAYS	{ } Command
Number of days between two dates date ₁ date ₂ → Δdays		
<i>Related Flag: -42</i>		

DEPND  [PLOT] PPAR DEPND	Command
Specifies plot dependent column, variable, or range n → ‘name’ → { name } → start end → { start end } → { name start end } → { name [initial values] tolerance } →	
DEPTH  [STACK] DEPTH	Command
Counts the objects on the stack objects → objects n	
DET [MTH] MATR NORM [NXT] DET { } Command	
Determinant of a square matrix [[matrix]] → determinant	
DETACH  [LIBRARY] DETAC	Command
Detaches library from current directory library-number →	
→DIAG [MTH] MATR [NXT] →DIAG G { } Command	
Returns vector of major diagonal elements [[matrix]] → [diagonals]	
DIAG→ [MTH] MATR [NXT] DIAG+ G Command	
Creates matrix with specified diagonal elements [diagonals] n → [[nXn matrix]] [diagonals] { m n } → [[mXn matrix]]	
DIFFEQ  [PLOT] PTYPE DIFFE	G Command
Selects differential equation plot	
DISP [PRG] [NXT] OUT DISP { } Command	
Displays an object in medium font (5x7) on line <i>n</i> , where <i>n</i> =1 is the top line, <i>n</i> =7 is the bottom line object n →	
DO [PRG] BRCH DO DO	Command
Begins DO loop DO loop-clause UNTIL test-clause END	

DOERR	[PRG] [NXT] ERROR DOERR	Command
Generates system or user-defined error		
	0 →	<i>Simulates [CANCEL]</i>
	n →	<i>Issues machine error n</i>
	#n →	<i>Issues machine error n</i>
	"string" →	<i>Issues string error</i>
DOLIST	[PRG] LIST PROC DOLIS	G Command
Applies <i>n</i> arguments in lists to <i>object</i> . If <i>object</i> is a command, a program containing one command, or a user-defined function then <i>n</i> can be omitted.		
	{ list ₁ } ... { list _n } n object →	{ results }
	{ list ₁ } ... { list _n } command →	{ results }
DOSUBS	[PRG] LIST PROC DOSUB	G Command
Executes a program or command using arguments in a list		
	{ list } n « program » →	{ list }
	{ list } n command →	{ list }
	{ list } n 'name' →	{ list }
	{ list } « program » →	{ list }
	{ list } command →	{ list }
	{ list } name →	{ list }
DOT	[MTH] VECTR DOT	{ } Command
Dot product of two vectors		
	[vector A] [vector B] →	x
DRAW	[←] [PLOT] DRAW	Command
Draws a plot		
<i>Related Flags: -30, -31</i>		
DRAX	[←] [PLOT] DRAX	Command
Draws axes		
DROP	[←] [DROP]	Command
Drops one object off the stack		
	object →	
DROPN	[←] [STACK] [NXT] DRPN	Command
Drops <i>n</i> and <i>n</i> objects from the stack		
	obj _n ... obj ₁ n →	
DROP2	[←] [STACK] [NXT] DROP2	Command
Drops two objects from the stack		
	obj ₂ obj ₁ →	




DTAG	[PRG] TYPE [NXT] DTAG	Command
Removes all tags from object :tag:obj → obj		
DUP	[↵] [STACK] [NXT] DUP	Command
Duplicates one object on the stack obj → obj obj		
DUPN	[↵] [STACK] [NXT] DUPN	Command
Duplicates n objects on the stack (excluding n) obj _{n} ... obj ₁ n → obj _{n} ... obj ₁ obj _{n} ... obj ₁		
DUP2	[↵] [STACK] [NXT] DUP2	Command
Duplicates two objects on the stack obj ₁ obj ₂ → obj ₁ obj ₂ obj ₁ obj ₂		
D→R	[MTH] REAL [NXT] [NXT] D→R	{ } Command
Degrees-to-radians conversion $x \rightarrow (\pi/180)x$ 'symb' → 'D→R(symb)' <i>Related Flag: -3</i>		
e	[α] [↵] [E] [MTH] [NXT] CONS E] Function
Symbolic constant e → 2.71828182846 → 'e' <i>Related Flags: -2, -3</i>		
EGV	[MTH] MATR [NXT] EGV	G { } Command
Computes eigenvalues and right eigenvector of a square matrix [[matrix]] → [[eigenvector]] [eigenvalues]		
EGVL	[MTH] MATR [NXT] EGVL	G { } Command
Computes eigenvalues of a square matrix [[matrix]] → [eigenvalues]		
ELSE	[PRG] BRCH IF ELSE [PRG] [NXT] ERROR IFERR ELSE	Command
Begins false-clause in IF or IFERR structures		
END	[PRG] BRCH IF END [PRG] BRCH CASE END [PRG] BRCH DO END [PRG] BRCH WHILE END [PRG] [NXT] ERROR IFERR END	Command
Ends program structures		

ENDSUB	[PRG]	LIST PROC ENDS	G Command
Returns the number of frames in argument list for DOSUBS → n			
ENG	[MODES]	FMT ENG	Command
Sets Engineering display mode n →			
EQ→	[PRG]	TYPE [NXT] EQ+	{ } Command
Separates equation into left and right sides ‘symb ₁ =symb ₂ ’ → ‘symb ₁ ’ ‘symb ₂ ’ z → z 0 ‘name’ → ‘name’ 0 x_unit → x_unit 0 ‘symb’ → ‘symb’ 0			
EQNLIB	[EQLIB]	EQLIB EQNLI	G Command
Displays the Equation Library Catalog			
ERASE	[PLOT]	ERASE	Command
Erases <i>PICT</i>			
ERRM	[PRG]	[NXT] ERROR ERRM	Command
Returns the last error message → “error message”			
ERRN	[PRG]	[NXT] ERROR ERRN	Command
Returns the last error number. Returns #70000h if the last error was generated by DOERR with a string argument. → #n			
ERRO	[PRG]	[NXT] ERROR ERRO	Command
Clears the last error number			
EVAL	[EVAL]		Command
Evaluates an object obj → :port:name → :port:{path name} → { port:name ₁ port:name ₂ ... } → <i>Related Flag:</i> -3			
EXP	[e^x]		{ } ↓ ∂ } Function
Natural exponential z → exp z ‘symb’ → ‘EXP(symb)’ <i>Related Flag:</i> -3			

EXPAN	 SYMBOLIC	EXPA	{ } Command
Expands an algebraic			
$z \rightarrow z$ $\text{'symb}_1 \rightarrow \text{'symb}_2$			
EXPFIT	 STAT	ΣPAR MODL EXPFI	Command
Selects exponential curve-fitting model			
EXPM	MTH	HYP NXT EXPM	{ } ↓ ∂ Function
Natural exponential minus 1			
$x \rightarrow \exp(x)-1$ $\text{'symb'} \rightarrow \text{'EXPM(symb)'}$			
<i>Related Flag: -3</i>			
EYEPT	 PLOT NXT	3D VPAR	G Command
NXT EYEPT			
Specifies the coordinates of the eye-point in a 3D plot			
$x_{eye} \ y_{eye} \ z_{eye} \rightarrow$			
<i>Note: y_{eye} must be 1 unit less than y_{near}</i>			
F0λ	 EQ LIB	UTILS F0λ	G { } Function
Calculates fraction of black-body emissive power at temperature T between wavelengths 0 and λ. If units are not supplied, λ is assumed to be meters and T is assumed to be K.			
$\lambda \ T \rightarrow \text{fraction}$ $\text{'symb'} \ T \rightarrow \text{'F0}\lambda(\text{symb},T)'$ $\lambda \ \text{'symb'} \rightarrow \text{'F0}\lambda(\lambda,\text{symb})'$ $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'F0}\lambda(\text{symb}_1,\text{symb}_2)'$			
<i>Related Flag: -3</i>			
FACT			{ } Function
Factorial or gamma function			
$n \rightarrow n!$ $x \rightarrow \Gamma(x+1)$ $\text{'symb'} \rightarrow \text{'FACT(symb)'}$			
<i>Related Flag: -3</i>			
FANNING	 EQ LIB	UTILS FANNI	G { } Function
Calculates Fanning friction factor			
$e/D \ Re \rightarrow f$ $\text{'symb'} \ x \rightarrow \text{'FANNING(symb,x)'}$ $x \ \text{'symb'} \rightarrow \text{'FANNING(x,symb)'}$ $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'FANNING(symb}_1,\text{symb}_2)'$			
<i>Related Flag: -3</i>			

FC?	(PRG) TEST (NXT) (NXT) FC?	{ } Command
Tests a system or user flag		
$\pm n \rightarrow T/F$		
FC?C	(PRG) TEST (NXT) (NXT) FC?C	{ } Command
Tests and clears a system or user flag		
$\pm n \rightarrow T/F$		
FFT	(MTH) (NXT) FFT FFT	G { } Command
Discrete Fourier transform		
[vector] \rightarrow [vector]'		
[[matrix]] \rightarrow [[matrix]]'		
<i>Note: Array dimensions must be a power of 2</i>		
FINDALARM	(TIME) ALRM FINDA	Command
Returns alarm index n or 0 if no alarm is found		
<i>First alarm due after a date and time:</i>		
{ date time } $\rightarrow n$		
<i>First alarm due on a specified date:</i>		
date $\rightarrow n$		
<i>First past due alarm:</i>		
0 $\rightarrow n$		
FINISH	(I/O) SRVR FINIS	Command
Terminates Kermit server mode		
<i>Related Flags: -33, -39</i>		
FIX	(MODES) FMT FIX	Command
Sets Fix display mode		
$n \rightarrow$		
FLOOR	(MTH) REAL (NXT) (NXT) FLOOR	{ } Function
Next smallest integer		
$x \rightarrow n$		
'symb' \rightarrow 'FLOOR(symb)'		
$x_unit \rightarrow n_unit$		
<i>Related Flag: -3</i>		
FOR	(PRG) BRCH FOR FOR	Command
Begins FOR structure		
start end FOR counter loop-clause NEXT		
start end FOR counter loop-clause increment STEP		

FP	[MTH] REAL [NXT] FP	{ } Function
Fractional part		
	x → y	
	'symb' → 'FP(symb)'	
	x_unit → y_unit	
<i>Related Flag: -3</i>		
FREE		Command
Frees (makes independent) merged memory, moving specified objects to the newly-independent RAM card. Retained for HP 48SX compatibility.		
	{ } port →	
	LID port →	
	:port:name port →	
	{ :port:names ... LIDs } port →	
<i>Note: Ports 1 & 2 may be merged in the HP 48SX, but only port 1 may be merged in the HP 48GX.</i>		
FREE1	[←] [LIBRARY] FREE1	G Command
Frees (makes independent) merged memory, moving specified objects to the newly-independent RAM card in port 1		
	{ } →	
	LID →	
	:port:name →	
	{ :port:names ... LIDs } →	
FREEZE	[PRG] [NXT] OUT FREEZ	{ } Command
Freezes up to three display areas until a key is pressed. The least significant bits of <i>n</i> control which areas will be frozen.		
	n →	
	Bit: 0	Status area
	1	Stack & command line
	2	Menu area
FS?	[PRG] TEST [NXT] [NXT] FS?	{ } Command
	[←] [MODES] FLAG FS?	
Tests a system or user flag		
	±n →	T/F
FS?C	[PRG] TEST [NXT] [NXT] FS?C	{ } Command
	[←] [MODES] FLAG FS?C	
Tests and clears a system or user flag		
	±n →	T/F

FUNCTION  [PLOT] PTYPE FUNC	Command
Selects function plot	
GET [PRG] LIST ELEM GET	Command
Gets an element from a list, vector, or matrix	
{ list } position → object	
'name' position → object	
[vector] position → z	
[[matrix]] position → z	
[[matrix]] { row col } → z	
'name' { row col } → z	
<i>Note:</i> Position arguments for matrices are specified in row order	
GETI [PRG] LIST ELEM GETI	Command
Gets an element from a list, vector, or matrix, increments and returns the position, and returns the list. Sets flag -64 if the position wraps to the first element, otherwise clears flag -64.	
{ list } position → { list } position' object	
'name' position → 'name' position' object	
[vector] position → [vector] position' z	
[[matrix]] position → [[matrix]] position' z	
[[matrix]] { row col } → [[matrix]] { row col' } z	
'name' { row col } → 'name' { row col' } z	
<i>Related flag:</i> -64	
<i>Note:</i> Position arguments for matrices are specified in row order	
GOR [PRG] GROB GOR	Command
Superimposes grob' onto grob at the specified coordinates	
grob (x,y) grob' → grob''	
grob { #x #y } grob' → grob''	
PICT (x,y) grob' →	
PICT {#x #y } grob' →	
GRAD  [MODES] ANGL GRAD	Command
Sets Grads mode	
GRAPH	Command
Enters the graphics environment until [CANCEL] is pressed	
GRIDMAP  [PLOT] [NXT] 3D PTYPE GRID G	Command
Selects the gridmap plot type	

→GROB	[PRG] GROB →GRO	{ } Command
Converts object into graphics object		
	object n → grob	
	0	<i>EquationWriter picture</i>
	1	<i>Small font (3x5)</i>
	2	<i>Medium font (5x7)</i>
	3	<i>Large font (5x9)</i>
<i>Related Flags: -45 through -51</i>		
GXOR	[PRG] GROB GXOR	Command
Superimposes and inverts grob' onto grob at the specified coordinates		
	grob (x,y) grob' → grob''	
	grob { #x #y } grob' → grob''	
	PICT (x,y) grob' →	
	PICT {#x #y } grob' →	
*H	[↵] PLOT PPAR [NXT] *H	Command
Multiplies the vertical plot scale by specified factor (alters PPAR)		
	x →	
HALT	[PRG] [NXT] RUN HALT	Command
Suspends program execution until either CONT or KILL are executed		
HEAD	[↵] CHARS [NXT] HEAD	G Command
	[PRG] LIST ELEM [NXT] HEAD	
Returns the first object in a list or the first character in a string		
	{ obj ₁ ... obj _n } → obj ₁	
	"ABCD" → "A"	
HEX	[MTH] BASE HEX	Command
Sets hexadecimal base		
<i>Related Flags: -5 through -12</i>		
HISTOGRAM	[↵] PLOT [NXT] STAT PTYPE	Command
	HISTO	
Selects histogram plot		
HISTPLOT	[↵] STAT PLOT HISTP	Command
Draws a histogram of the data in ΣDAT		
HMS+	[↵] TIME [NXT] HMS+	{ } Command
Adds in HH.MMSSs format		
	hms ₁ hms ₂ → hms ₁ + hms ₂	

HMS-	(TIME) (NXT) HMS-	{ } Command
Subtracts in HH.MMSSs format $\text{hms}_1 \text{ hms}_2 \rightarrow \text{hms}_1 - \text{hms}_2$		
HMS→	(TIME) (NXT) HMS→	{ } Command
Converts a number from HH.MMSSs format $\text{hms} \rightarrow x$		
→HMS	(TIME) (NXT) →HMS	{ } Command
Converts a number to HH.MMSSs format $x \rightarrow \text{hms}$		
HOME	(HOME)	Command
Selects the <i>HOME</i> directory		
i	 (MTH) (NXT) CONS I	∂ Function
Symbolic constant <i>i</i> $\rightarrow (0,1)$ $\rightarrow 'i'$ <i>Related Flags:</i> -2, -3		
IDN	(MTH) MATR MAKE IDN	{ } Command
Creates an identity matrix $n \rightarrow [[n \times n \text{ real-identity-matrix}]]$ $[[\text{matrix}]] \rightarrow [[\text{identity-matrix}]]$ 'name' \rightarrow <i>Replaces named matrix</i>		
IF	(PRG) BRCH IF IF	Command
Begins IF structure IF test THEN true-clause END IF test THEN true-clause ELSE false-clause END		
IFERR	(PRG) (NXT) ERROR IFERR IFERR	Command
Begins IFERR structure IFERR trap-clause THEN error-clause END IFERR trap-clause THEN error-clause ELSE normal-clause END <i>Related Flag:</i> -55		
IFFT	(MTH) (NXT) FFT IFFT	G { } Command
Inverse discrete Fourier transform $[\text{vector}] \rightarrow [\text{vector}]'$ $[[\text{matrix}]] \rightarrow [[\text{matrix}]]'$ <i>Note:</i> Array dimensions must be a power of 2		






IFT	(PRG) BRCH (NXT) IFT	Command
IF ... THEN ... END test. Executes <i>object</i> if T/F is true. T/F object →		
IFTE	(PRG) BRCH (NXT) IFTE	∂ Function
IF ... THEN ... ELSE ... END test. Executes <i>true-obj</i> if T/F is true T/F true-obj false-obj → 'symb' true-obj false-obj → 'IFTE(symb,true-obj,false-obj)'		
IM	(MTH) (NXT) CMPL IM	{ } Function
Returns imaginary part of a number or array x → 0 (x,y) → y [R-array] → [zero R-array] [C-array] → [R-array] 'symb' → 'IM(symb)'		
<i>Related Flag: -3</i>		
INCR	(G) (MEMORY) ARITH INCR	{ } Command
Increments then returns the value of a variable 'name' → x		
INDEP	(G) (PLOT) PPAR INDEP	Command
Specifies plot independent column, variable, or range n → 'name' → { name } → start end → { start end } → { name start end } →		
INFORM	(PRG) (NXT) IN INFOR	G Command
Displays an input form and waits for user input. INFORM terminates when the user presses CANCEL , CANCL , or OK . For further details, see <i>Data Entry</i> . "title" {labels} format {defaults} {initial_values} → {data} 1 "title" {labels} format {defaults} {initial_values} → 0 <i>cancelled</i>		






INPUT	[PRG] [NXT]	IN INPUT	Command
Suspends program, displays message, and waits for data. The <i>modes</i> can be one or more of: ALG (algebraic/program-entry mode), α (alpha lock), or V (verify syntax). If <i>column</i> is 0 it specifies the end of the row. A positive value for <i>row</i> (or <i>column</i> if only the column is specified) displays the insert cursor; a negative value specifies the replace cursor. The level 1 list may contain any of these options in any order. For further details, see <i>Data Entry</i> .			
“message” “prompt” → “result” “message” { mode(s) } → “result” “message” { “prompt” column } → “result” “message” { “prompt” mode(s) } → “result” “message” { “prompt” column mode(s) } → “result” “message” { “prompt” { row column } mode(s) } → “result”			
INV	[$\frac{1}{x}$]	{ } $\downarrow \partial$ }	Function
Inverse (reciprocal)			
$z \rightarrow 1/z$ [[matrix]] → [[1/matrix]] ‘symb’ → ‘INV(symb)’ x_unit → 1/x_1/unit			
<i>Related Flag:</i> -3			
IP	[MTH] REAL [NXT]	IP	{ } Function
Integer part			
$x \rightarrow n$ ‘symb’ → ‘IP(symb)’ x_unit → n_unit			
<i>Related Flag:</i> -3			
ISOL	[\leftarrow] [SYMBOLIC]	ISOL	{ } Command
Isolates a variable in an equation			
‘symb ₁ ’ ‘global’ → ‘symb ₂ ’			
<i>Related Flags:</i> -1, -3			
KERRM	[\leftarrow] [I/O] [NXT]	KERR	Command
Returns the last Kermit error message			
→ “message”			











KEY	[PRG] [NXT] IN KEY	Command
Returns 0 if no key has been pressed, otherwise 1 in level 1 and the keycode in level 2		
→ 0		
→ rc 1		
KGET	[↵] [I/O] SRVR KGET	Command
Gets named data from a remote device		
‘name’ →		
“name” →		
{ remote-name local-name } →		
{ name ₁ name ₂ ... } →		
{ { remote-name ₁ local-name ₁ } name ₂ ... } →		
<i>Related Flags: -33, -35, -36, -39</i>		
KILL	[PRG] [NXT] RUN KILL	Command
Cancels all suspended programs		
LABEL	[↵] [PLOT] [NXT] LABEL	Command
Labels axes		
<i>Related Flag: -30</i>		
LAST	[PRG] [NXT] ERROR LASTA	Command
LASTARG		
Returns arguments (saved if flag -55 is clear)		
→ Last_Arguments		
<i>Related Flag: -55</i>		
LCD→	[PRG] GROB [NXT] LCD→	Command
Returns LCD as 131x64 pixel graphics object		
→ grob		
→LCD	[PRG] GROB [NXT] →LCD	Command
Displays graphics object at the upper-left corner of the stack display		
grob →		
LIBEVAL	G Command	
Executes library object given its six-digit (hex) binary integer reference. The upper three digits are the library number and the lower three digits are the function number in the library.		
#n →		
LIBS	[↵] [LIBRARY] LIBS	Command
Lists library objects attached to current directory		
→ { “title ₁ ” library-number ₁ port ₁ ... }		

LINE	(PRG) PICT LINE	Command
Draws a line in <i>PICT</i> between two coordinates $(x_1, y_1) (x_2, y_2) \rightarrow$ $\{ \#x_1 \#y_1 \} \{ \#x_2 \#y_2 \} \rightarrow$		
ΣLINE	(\rightarrow) (STAT) FIT Σ LINE	Command
Returns best-fit line for data in Σ DAT with values for a and b filled in <i>Linear model</i> \rightarrow 'a+b*X' <i>Logarithmic model</i> \rightarrow 'a+b*LN(X)' <i>Exponential model</i> \rightarrow 'a*EXP(b*X)' <i>Power model</i> \rightarrow 'a*X^b'		
LINFIT	(\rightarrow) (STAT) Σ PAR MODL LINFIT	Command
Selects linear curve-fitting model		
LININ	(PRG) TEST (NXT) (NXT) (NXT) LININ	G Command
Determines whether an equation is linear in a variable 'symb' 'name' \rightarrow T/F		
LIST\rightarrow		Command
Separates a list into individual objects $\{ \text{obj}_1 \dots \text{obj}_n \} \rightarrow \text{obj}_1 \dots \text{obj}_n \ n$		
\rightarrowLIST	(PRG) LIST \rightarrow LIST	Command
Combines n objects into a list $\text{obj}_1 \dots \text{obj}_n \ n \rightarrow \{ \text{obj}_1 \dots \text{obj}_n \}$		
ΔLIST	(MTH) LIST LIST	G Command
Computes the first differences of objects in a list $\{ \text{obj}_1 \text{obj}_2 \text{obj}_3 \dots \text{obj}_n \} \rightarrow \{ \text{obj}_{2-1} \text{obj}_{3-2} \dots \}$		
ΣLIST	(MTH) LIST Σ LIST	G Command
Sums the elements in a list $\{ \text{obj}_1 \text{obj}_2 \dots \text{obj}_n \} \rightarrow \text{obj}_1 + \text{obj}_2 + \dots + \text{obj}_n$ <i>Related Flags:</i> -5 through -10		
\prodLIST	(MTH) LIST π LIST	G Command
Returns the product of elements in a list $\{ \text{obj}_1 \text{obj}_2 \dots \text{obj}_n \} \rightarrow \text{obj}_1 * \text{obj}_2 * \dots * \text{obj}_n$ <i>Related Flags:</i> -5 through -10		
LN	(\rightarrow) (LN)	{ } \downarrow ∂ } Function
Natural logarithm $z \rightarrow \ln z$ 'symb' \rightarrow 'LN(symb)' <i>Related Flags:</i> -1, -3, -22		

LNP1	[MTH]	HYP	[NXT]	LNP1	{ } ↓ ∂ Function
Natural logarithm of (argument + 1)					
x → ln(1+x)					
'symb' → 'LNP1(symb)'					
<i>Related Flags:</i> -3, -22					
LOG	[→]	[LOG]			{ } ↓ ∂ Function
Common (base 10) logarithm					
z → log z					
'symb' → 'LOG(symb)'					
<i>Related Flags:</i> -1, -3, -22					
LOGFIT	[↵]	[STAT]	ΣPAR	MODL	LOGFI Command
Selects logarithmic curve-fitting model					
LQ	[MTH]	MATR	FACTR	LQ	G { } Command
Returns the LQ factorization of a matrix					
[[matrix A]] → [[matrix L]] [[matrix Q]] [[matrix P]]					
LR	[↵]	[STAT]	FIT	LR	Command
Computes linear regression of ΣDAT data					
→ intercept slope					
LSQ	[MTH]	MATR	LSQ		G { } Command
	[↵]	[SOLVE]	SYS	LSQ	
Returns the minimum norm least squares solution to under- or over-determined system of linear equations AX=B.					
[vector B] [[matrix A]] → [vector X]					
[[matrix B]] [[matrix A]] → [[matrix X]]					
<i>Related Flag:</i> -54					
LU	[MTH]	MATR	FACTR	LU	G Command
Returns the Crout LU decomposition of a square matrix					
[[matrix A]] → [[matrix L]] [[matrix U]] [[matrix P]]					
MANT	[MTH]	REAL	[NXT]	MANT	{ } Function
Returns the mantissa of a number					
x → y					
'symb' → 'MANT(symb)'					
<i>Related Flag:</i> -3					
↑MATCH	[↵]	[SYMBOLIC]	[NXT]	↑MAT	Command
Match-and-replace, beginning with subexpressions					
'symb' { 'pattern' 'replacement' } → 'result' T/F					
'symb' { 'pattern' 'replacement' 'conditional' } → 'result' T/F					









↓MATCH  [SYMBOLIC] [NXT] ↓MAT	Command
Match-and-replace, beginning with top-level expression 'symb' { 'pattern' 'replacement' } → 'result' T/F 'symb' { 'pattern' 'replacement' 'conditional' } → 'result' T/F	
MAX [MTH] REAL MAX { }	Function
Returns the maximum of two numbers $x \ y \rightarrow \max(x,y)$ $x \text{ 'symb' } \rightarrow \text{'MAX'(x,symb)}$ 'symb' $x \rightarrow \text{'MAX'(symb,x)}$ 'symb ₁ ' 'symb ₂ ' → 'MAX(symb ₁ ,symb ₂)' $x_unit \ y_unit \rightarrow \max(x,y)_unit$	
<i>Related Flag: -3</i>	
MAXR [MTH] [NXT] CONS [NXT] MAXR ∂	Function
Symbolic constant – maximum HP 48 real number → 9.999999999999E499 → 'MAXR'	
<i>Related Flags: -2, -3</i>	
MAXΣ  [STAT] IVAR MAXΣ	Command
Finds the maximum column values of the data in ΣDAT → x → $[x_1 \dots x_m]$	
MCALC  [EQLIB] MES MCAL $G \{ \}$	Command
Sets Multiple Equation Solver variable to <i>not</i> user-defined state 'name' → "ALL" →	
<i>Related Flag: 63</i>	
MEAN  [STAT] IVAR MEAN	Command
Computes means of the data in ΣDAT → x → $[x_1 \dots x_m]$	
MEM  [MEMORY] MEM	Command
Performs garbage collection and returns available memory (see <i>Temporary Memory</i>) → x	

MENU	 MODES	MENU MENU	Command
Selects a built-in menu or creates a custom menu (see <i>Built-in Menus</i>)			
		mm.pp →	
		'list-name' →	
		{ names and commands } →	
MERGE			Command
Merges RAM card with main memory. Retained for HP 48SX compatibility.			
		port →	
<i>Note:</i> Ports 1 & 2 may be merged in the HP 48SX, but only port 1 may be merged in the HP 48GX.			
MERGE1	 LIBRARY	MERG	G Command
Merges RAM card in port 1 with main memory			
MIN	MTH	REAL MIN	{ } Function
Returns the minimum of two numbers			
		x y → min(x,y)	
		x 'symb' → 'MIN(x,symb)'	
		'symb' x → 'MIN(symb,x)'	
		'symb ₁ ' 'symb ₂ ' → 'MIN(symb ₁ ,symb ₂)'	
		x_unit y_unit → min(x,y)_unit	
<i>Related Flag:</i> -3			
MINEHUNT	 EQ LIB	UTILS MINE	G Command
Starts the Minehunt game			
MINIT	 EQ LIB	MES MINIT	G Command
Establishes <i>Mpar</i> from <i>EQ</i>			
MINR	MTH NXT	CONS NXT MINR	∂ Function
Symbolic constant – minimum HP 48 real number			
		→ 1.E-499	
		→ 'MINR'	
<i>Related Flags:</i> -2, -3			
MINΣ	 STAT	IVAR MINΣ	Command
Finds the minimum column values of the data in Σ <i>DAT</i>			
		→ x	
		→ [x ₁ ... x _m]	






MITM  EQ LIB MES MITM	G Command
Changes the title and variable menu in <i>Mpar</i> “title” { name ₁ ... name _n } → <i>Note:</i> Use “” to include a blank label.	
MOD  MTH REAL MOD	{ } Function
Modulo $x \ y \rightarrow x \bmod y$ $x \text{ 'symb' } \rightarrow \text{'MOD'(x,symb)}$ $\text{'symb' } x \rightarrow \text{'MOD(symb,x)}$ $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'MOD(symb}_1,\text{symb}_2)}$	
<i>Related Flag:</i> -3	
MROOT  EQ LIB MES MROO	G Command
Solves for single or all variables using the Multiple Equation Solver “name” → value “ALL” →	
MSGBOX  PRG  NXT OUT MSGB	G { } Command
Displays a message (up to 75 characters) and waits for acknowledgement keystroke “message” →	
MSOLVR  EQ LIB MES MSOL	G Command
Displays the Multiple Equation Solver menu <i>Related Flag:</i> 63	
MUSER  EQ LIB MES MUSE	G { } Command
Sets Multiple Equation Solver variable to user-defined state “name” → “ALL” →	
<i>Related Flag:</i> 63	
NDIST  MTH  NXT PROB  NXT NDIST	G { } Command
Normal probability density mean variance x → ndist(mean, variance, x)	










NEG		{ } ↓ ∂ Function
CMPL NEG		
Negates an argument		
	$z \rightarrow -z$	
	$\#n_1 \rightarrow \#n_2$ (<i>two's complement</i>)	
	$x_unit \rightarrow -x_unit$	
	$[vector] \rightarrow [-vector]$	
	$[[matrix]] \rightarrow [[-matrix]]$	
	$'symb' \rightarrow -(symb)'$	
	$grob \rightarrow inverted_grob$	
	$PICT \rightarrow inverts\ PICT$	
<i>Related Flags:</i> -3, -5 through -10		
NEWOB	NEWO	Command
Separates object from list or backup object (see <i>Temporary Memory</i>)		
	$object \rightarrow object$	
<i>Related Flag:</i> -55 (Should be set if freeing an object from a list)		
NEXT	BRCH START BRCH FOR NEXT	Command
Ends FOR ... NEXT or START ... NEXT		
NOT	BASE LOGIC NOT TEST NOT	{ } Function
Logical or binary NOT		
	$\#n_1 \rightarrow \#n_2$	
	$x \rightarrow T/F$	
	$'symb' \rightarrow 'NOT(symb)'$	
	$"string_1" \rightarrow "string_2"$	
<i>Related Flags:</i> -3, -5 through -10		
NOVAL	IN NOVA	G Command
Placeholder for unspecified values in INFORM field data list		
	$\rightarrow NOVAL$	
<i>Note:</i> HP 48 operating system versions K and L contain an error – NOVAL <i>doesn't</i> place itself on the stack. If you're writing a program to be distributed to a wide audience, it may be best to use the sequence { NOVAL } HEAD when you need to place NOVAL on the stack.		
NSUB	LIST PROC NSUB	G Command
Returns the current frame number for DOSUBS		
	$\rightarrow n$	

NUM	CHARS NUM { } Command	
	PRG TYPE NXT NUM	
	Returns character code of a string's first character "string" → n	
→NUM	→NUM Command	
	Evaluates an object to yield a numeric result object → z	
NUMX	PLOT NXT 3D VPAR G Command	
	NXT NUMX	
	Specifies number of plot increments in X for 3D plot types (must be ≥ 2) n →	
NUMY	PLOT NXT 3D VPAR G Command	
	NXT NUMY	
	Specifies number of plot increments in Y for 3D plot types (must be ≥ 2) n →	
NΣ	STAT SUMS NΣ Command	
	Returns the number of data points in ΣDAT → n	
OBJ→	PRG TYPE OBJ→ Command	
	CHARS NXT OBJ→	
	Decomposes an object into individual components. String objects are executed as a command line after the “ ” delimiters have been removed.	
	:tag:object → object "tag" (x,y) → x y x_units → x 1_units 'X+Y' → X Y 2 + [x ₁ ... x _n] → x ₁ ... x _n n [[x ₁₁ x ₁₂ ... x _{nm}]] → x ₁₁ ... x _{nm} { n m } { obj ₁ ... obj _n } → obj ₁ ... obj _n n "string" →	
OCT	MTH BASE OCT Command	
	Sets octal base <i>Related Flags:</i> -5 through -12	
OFF	PRG NXT OFF Command	
	Turns the calculator off	








OLDPR  [I/O] PRINT PRTPA OLDPR	Command
Remaps printer output to the HP 82240A character set	
OPENIO  [I/O] [NXT] SERIA OPENI	Command
Opens IR port or wired serial port <i>Related Flag:</i> -33	
OR [MTH] BASE [NXT] LOGIC OR { } Function [PRG] TEST [NXT] OR	
Logical or binary OR $\begin{aligned} \#n_1 \#n_2 &\rightarrow \#n_3 \\ x \ y &\rightarrow T/F \\ x \text{ 'symb' } &\rightarrow \text{'x OR symb' } \\ \text{'symb' } x &\rightarrow \text{'symb OR x' } \\ \text{'symb}_1 \text{ 'symb}_2 &\rightarrow \text{'symb}_1 \text{ OR symb}_2 \text{' } \\ \text{"string}_1 \text{ "string}_2 &\rightarrow \text{"string}_3 \text{"} \end{aligned}$	
<i>Related Flags:</i> -3, -5 through -10	
ORDER  [MEMORY] DIR ORDER	Command
Rearranges the VAR menu $\{ \text{names} \} \rightarrow$	
OVER  [STACK] OVER	Command
Copies the object in level 2 into level 1 $\text{obj}_2 \text{ obj}_1 \rightarrow \text{obj}_2 \text{ obj}_1 \text{ obj}_2$	
PARAMETRIC  [PLOT] PTYPE PARA	Command
Selects parametric plot	
PARITY  [I/O] IOPAR PARIT	Command
Sets parity. n<0 indicates transmit parity only $\begin{aligned} n &\rightarrow \\ 0 &\text{ none} \\ 1 &\text{ odd} \\ 2 &\text{ even} \\ 3 &\text{ mark} \\ 4 &\text{ space} \end{aligned}$	
PARSURFACE  [PLOT] [NXT] 3D PTYPE G Command PARSU	
Selects parametric surface plot	
PATH  [MEMORY] DIR PATH	Command
Returns a list showing the current path $\rightarrow \{ \text{HOME directory-names} \}$	

PCOEF	[F4] [SOLVE] POLY PCOEF	G { }	Command
Computes coefficients of a polynomial with specified roots [roots] → [coefficients]			
PCONTOUR	[F4] [PLOT] [NXT] 3D PTYPE	G	Command
PCON Selects the pcontour plot type			
PCOV	[F4] [STAT] FIT [NXT] PCOV	G ↓ ∂	Function
Population covariance of Σ DATA data in columns specified by COLΣ → covariance			
PDIM	[PRG] PICT PDIM		Command
Changes the size of PICT (x_{\min}, y_{\min}) (x_{\max}, y_{\max}) → <i>Changes PICT relative to the current user coordinates</i> #horizontal #vertical → <i>Does not affect current user coordinates</i> <i>Note: The width of PICT cannot exceed 2048 pixels</i>			
PERM	[MTH] [NXT] PROB PERM	{ }	Function
Permutations of n objects taken m at a time $n \ m \rightarrow P_{n,m}$ 'symb' $m \rightarrow$ 'PERM(symb,m)' n 'symb' → 'PERM(n,symb)' 'symb ₁ ' 'symb ₂ ' → 'PERM(symb ₁ ,symb ₂)' <i>Related Flag: -3</i>			
PEVAL	[F4] [SOLVE] POLY PEVAL	G { }	Command
Evaluates polynomial with specified coefficients at x [coefficients] $x \rightarrow y$			
PGDIR	[F4] [MEMORY] DIR PGDIR	{ }	Command
Purges specified directory and its contents 'name' →			
PICK	[F4] [STACK] PICK		Command
Copies n th object into level 1 (excluding n) $obj_n \dots obj_1 \ n \rightarrow obj_n \dots obj_1 \ obj_n$			
PICT	[PRG] PICT PICT		Command
Returns the name PICT to level 1 → PICT			
PICTURE	[F4] [PICTURE]	G	Command
Enters the graphics environment until [CANCEL] is pressed			

PINIT	 LIBRARY NXT PINIT	G Command
Initializes a memory card in port 2		
PIXOFF	PRG PICT NXT PIXOF	Command
Turns off a pixel in <i>PICT</i>		
	(x,y) → { #x #y } →	
PIXON	PRG PICT NXT PIXON	Command
Turns on a pixel in <i>PICT</i>		
	(x,y) → { #x #y } →	
PIX?	PRG PICT NXT PIX?	Command
Tests a pixel in <i>PICT</i>		
	(x,y) → T/F { #x #y } → T/F	
PKT	 I/O SRVR PKT	Command
Sends commands to server		
	“contents” “type” → “response”	
<i>Related Flags:</i> -33, -35, -39		
PMAX		Command
Sets the upper-right plot coordinates		
	(x,y) →	
PMIN		Command
Sets the lower-left plot coordinates		
	(x,y) →	
POLAR	 PLoT PTYPE POLAR	Command
Selects polar plot		
POS	 CHARS POS PRG LIST ELEM POS	Command
Finds a substring in a string or finds an object in a list		
	“string” “substring” → n { list } obj → n	
PREDV		{ } Command
Predicted dependent variable value		
	x → predicted_value	
PREDX	 STAT FIT PREDX	{ } Command
Predicted independent variable value		
	y → predicted_value	

PREDY  [STAT] FIT PREDY { } Command
Predicted dependent variable value $x \rightarrow \text{predicted_value}$
PRLCD  [I/O] PRINT PRLCD Command
Prints an image of the display <i>Related Flags:</i> -33, -34
PROMPT [PRG] [NXT] IN [NXT] PROM Command
Displays prompt and halts program “prompt” \rightarrow
PROOT  [SOLVE] POLY PROOT G { } Command
Computes roots of polynomial with real or complex coefficients $[z_1 \dots z_n] \rightarrow [\text{roots}]$ <i>Related Flag:</i> -22
PRST  [I/O] PRINT PRST Command
Prints the stack <i>Related Flags:</i> -33, -34, -37, -38
PRSTC  [I/O] PRINT PRSTC Command
Prints the stack in compact format <i>Related Flags:</i> -33, -34, -37, -38
PRVAR  [I/O] PRINT PRVAR { } Command
Prints the name and contents of one or more variables “name” \rightarrow :port:name \rightarrow <i>Related Flags:</i> -33, -34, -37, -38
PR1  [I/O] PRINT [NXT] PR1 Command
Prints the level 1 object object \rightarrow object <i>Related Flags:</i> -33, -34, -37
PSDEV  [STAT] IWAR [NXT] PSDEV G Command
Computes population standard deviation of the data in ΣDAT $\rightarrow x$ $\rightarrow [x_1 x_2 \dots x_m]$
PURGE  [PURGE] { } Command
Purges one or more variables “global” \rightarrow :port:name \rightarrow LID \rightarrow PICT \rightarrow

PUT	[PRG] LIST ELEM PUT	Command
Replaces an element in an array or list		
{ list ₁ } position obj → { list ₂ }		
'name' position obj →		
[vector ₁] position z → [vector ₂]		
[[matrix ₁]] position z → [[matrix ₂]]		
[[matrix ₁]] { row col } z → [[matrix ₂]]		
'name' { row col } z →		
PUTI	[PRG] LIST ELEM PUTI	Command
Replaces an element in an array or list and increments and returns the position. Sets flag -64 if the position wraps to the first element, otherwise clears flag -64.		
{ list ₁ } position obj → { list ₂ } position'		
'name' position obj → 'name' position'		
[vector ₁] position z → [vector ₂] position'		
[[matrix ₁]] position z → [[matrix ₂]] position'		
[[matrix ₁]] { row col } z → [[matrix ₂]] { row col }'		
'name' { row col } z → 'name' { row col }'		
<i>Related flag: -64</i>		
<i>Note: Position arguments for matrices are specified in row order</i>		
PVAR	[↵] [STAT] 1VAR [NXT] PVAR	G Command
Population variances of Σ DATA data in columns specified by COL Σ		
→ x		
→ [x ₁ x ₂ ... x _m]		
PVARS	[↵] [LIBRARY] PVARS	Command
Returns list of backup objects and library objects and the type of memory (or amount of memory if independent RAM)		
port → { list } "ROM"		
port → { list } "SYSRAM"		
port → { list } bytes		
PVIEW	[PRG] PICT [NXT] PVIEW [PRG] [NXT] OUT PVIEW	Command
Displays <i>PICT</i> with the specified coordinate or pixel at the upper-left corner. An empty list displays <i>PICT</i> centered in the display, ready to scroll until [CANCEL] is pressed.		
(x,y) →		
{ #x #y } →		
{ } →		

PWRFIT  [STAT] Σ PAR MODL PWRFI	Command
Selects power curve-fitting model	
PX→C [PRG] PICT [NXT] PX→C	Command
Pixel to user-unit coordinate conversion $\{ \#x \#y \} \rightarrow (x,y)$	
<i>Note:</i> Scaling information is derived from PPAR	
→Q  [SYMBOLIC] [NXT] $\rightarrow Q$	Command
Converts numbers to fractional equivalent $x \rightarrow 'a/b'$ $(x,y) \rightarrow 'a/b+c/d*i'$ $'X+1.4' \rightarrow 'X+7/5'$	
<i>Note:</i> The display mode (such as 2 FIX) affects the result	
→Qπ  [SYMBOLIC] [NXT] $\rightarrow Q \pi$	Command
→Q after factoring out π $x \rightarrow 'a/b*\pi'$ $x \rightarrow 'a/b'$ $(x,y) \rightarrow 'a/b*\pi+c/d*\pi*i'$ $(x,y) \rightarrow 'a/b*\pi+c/d*i'$ $(x,y) \rightarrow 'a/b+c/d*\pi*i'$ $(x,y) \rightarrow 'a/b+c/d*i'$ $'(2.5,3.5)*X' \rightarrow '(5/2+7/2*i)*X'$	
<i>Note:</i> The display mode (such as 2 FIX) affects the result	
QR [MTH] MATR FACTR QR G { }	Command
Computes the QR factorization of a matrix $[[\text{matrix A}]] \rightarrow [[\text{matrix Q}]] [[\text{matrix R}]] [[\text{matrix P}]]$	
QUAD  [SYMBOLIC] QUAD { }	Command
Solves a quadratic polynomial $'\text{symb}_1' \text{ 'global' } \rightarrow '\text{symb}_2'$	
<i>Related Flag:</i> -1	
QUOTE  [SYMBOLIC] [NXT] [NXT] QUOT	Function
Returns argument expression unevaluated $'\text{symb}' \rightarrow '\text{symb}'$	
RAD  [RAD]	Command
 [MODES] ANGL RAD	
Sets Radians mode <i>Related Flags:</i> -17, -18	

RAND	(MTH) (NXT) PROB RAND	Command
Returns a random number in the range $0 \leq x \leq 1$ → x		
RANK	(MTH) MATR NORM (NXT) RANK	G { } Command
Estimates the rank of a rectangular matrix [[matrix]] → rank <i>Related Flag:</i> -54		
RANM	(MTH) MATR MAKE RANM	G Command
Replaces or creates a random matrix of specified dimensions whose elements are integers in the range $-9 \leq n \leq 9$. 0 is twice as likely to occur as other values. [[matrix]] → [[matrix]]' { rows cols } → [[matrix]]'		
RCEQ	(PLOT) (PLOT) EQ (PLOT) (NXT) 3D (PLOT) EQ	Command
Recalls the current contents of the variable EQ → obj		
RCI	(MTH) MATR ROW RCI	G { } Command
Multiplies elements in a row of a matrix by a factor [[matrix]] factor row → [[matrix]]'		
RCIJ	(MTH) MATR ROW RCIJ	G { } Command
Multiplies elements in row _i of a matrix by a factor and adds the result to elements in row _j [[matrix]] factor i j → [[matrix]]'		
RCL	(PLOT) (RCL)	Command
Recalls the contents of a variable or backup object 'name' → obj PICT → grob :port:name → obj :port:{path name } → obj		
RCLALARM	(TIME) ALARM RCLAL	{ } Command
Recalls alarm from alarm list n → { date time action repeat }		




RCLF	[MODES] FLAG [NXT] RCLF	Command
Returns a list containing two binary integers representing the system and user flags. The least significant bit of each binary integer represents flag 1.		
→ { #system #user }		
<i>Note:</i> The binary integer wordsize should be set to 64 bits		
RCLKEYS	[MODES] KEYS RCLK	Command
Lists user-key assignments. S indicates standard keys are active.		
→ { obj ₁ rc.p ₁ ... obj _n rc.p _n }		
→ { S obj ₁ rc.p ₁ ... obj _n rc.p _n }		
<i>Related Flags:</i> -61, -62		
RCLMENU	[MODES] MENU RCLM	Command
Recalls number and page of active menu		
→ mm.pp		
<i>Note:</i> Menu number assignments are different between the HP 48S/SX and the HP 48G/GX. See <i>Built-in Menus</i> .		
RCLΣ	[PLOT] [NXT] STAT DATA [ΣDAT] [STAT] DATA [ΣDAT]	Command
Recalls the current contents of the variable ΣDAT		
→ object		
RCWS	[MTH] BASE [NXT] RCWS	Command
Recalls the binary integer wordsize		
→ n		
<i>Related Flags:</i> -5 through -12		
RDM	[MTH] MATR MAKE RDM	Command
Redimensions a matrix. Extra elements are dropped, missing elements are padded with zeros.		
[vector ₁] { cols } → [vector ₂]		
[vector] { rows cols } → [matrix]		
[[matrix]] { cols } → [vector]		
[[matrix ₁]] { rows cols } → [[matrix ₂]]		
'name' { cols } →		
'name' { rows cols } →		
RDZ	[MTH] [NXT] PROB RDZ	Command
Sets the random number seed. 0 will use the system clock.		
x →		












RE	(MTH) (NXT) CMPL RE	{ } Function
Returns the real part of a complex number, array, or unit object		
	$x \rightarrow x$	
	$(x,y) \rightarrow x$	
	$[R\text{-array}] \rightarrow [R\text{-array}]$	
	$[C\text{-array}] \rightarrow [R\text{-array}]$	
	$'\text{symb}' \rightarrow 'RE(\text{symb})'$	
	$x_unit \rightarrow x$	
<i>Related Flag: -3</i>		
RECN	(F) (I/O) (NXT) RECN	Command
Receives file from remote Kermit and saves in an object named in level 1		
	$'\text{name}' \rightarrow$	
	$"\text{name}" \rightarrow$	
<i>Related Flags: -33, -35, -36, -39</i>		
RECT	(F) (MODES) ANGL RECT (MTH) VECTR (NXT) RECT	Command
Sets rectangular coordinate mode		
RECV	(F) (I/O) RECV	Command
Receives file from remote Kermit and saves in a sender-named object		
<i>Related Flags: -33, -35, -36, -39</i>		
REPEAT	(PRG) BRCH WHILE REPEAT	Command
Begins loop clause in WHILE ... REPEAT ... END		
	$T/F \rightarrow$	
REPL	(F) (CHARS) REPL (MTH) MATR MAKE (NXT) REPL (PRG) LIST REPL (PRG) GROB REPL	Command
Replaces the level 1 object onto the level 3 object at the location specified in level 2		
	$\{ \text{list} \} \ n \ \{ \text{sublist} \} \rightarrow \{ \text{list}' \}$	
	$"\text{string}" \ n \ " \text{substring}" \rightarrow " \text{string}"$	
	$\text{grob} \ (x,y) \ \text{subgrob} \rightarrow \text{grob}'$	
	$\text{grob} \ \{ \#x \ \#y \} \ \text{subgrob} \rightarrow \text{grob}'$	
	$PICT \ (x,y) \ \text{subgrob} \rightarrow$	
	$PICT \ \{ \#x \ \#y \} \ \text{subgrob} \rightarrow$	





RES	[PLOT] PPAR RES	Command
Sets the plot resolution in user-unit or pixel intervals		
n → Interval in user-units		
#n → Interval in pixels		
RESTORE	[MEMORY] [NXT] RESTO	Command
Replaces HOME directory with backup copy		
backup →		
:port:backup →		
REVLIST	[MTH] LIST REVL [PRG] LIST PROC REVL	G Command
Reverses the order of objects in a list		
{ obj ₁ ... obj _n } → { obj _n ... obj ₁ }		
RKF	[SOLVE] DIFFE RKF	G Command
Computes the solution of the initial value problem for a differential equation using the Runge-Kutta-Fehlberg (4,5) method		
{ t ₀ y ₀ f(t,y) } tolerance t _f → { t ₀ y ₀ f(t,y) } tolerance		
{ t ₀ y ₀ f(t,y) } { tolerance hsize } t _f → { t ₀ y ₀ f(t,y) } tolerance		
RKFERR	[SOLVE] DIFFE RKFE	G Command
Computes the change in solution and absolute error estimate for a specified step for a differential equation using the Runge-Kutta-Fehlberg (4,5) method		
{ t y f(t,y) } stepsize → { t y f(t,y) } stepsize Δy error		
RKFSTEP	[SOLVE] DIFFE RKFS	G Command
Computes the next solution step of the initial value problem for a differential equation within a given error tolerance using the Runge-Kutta-Fehlberg (4,5) method		
{ t y f(t,y) } tolerance stepsize → { t y f(t,y) } tolerance next-step		
RL	[MTH] BASE [NXT] BIT RL { }	Command
Rotates left by one bit		
#n ₁ → #n ₂		
Related Flags: -5 through -12		
RLB	[MTH] BASE [NXT] BYTE RLB { }	Command
Rotates left by one byte		
#n ₁ → #n ₂		
Related Flags: -5 through -12		

RND	(MTH)	REAL	(NXT)	(NXT)	RND	{ }	Function
Rounds a number to n decimal places if $0 \leq n \leq 11$, to n significant digits if $-11 \leq n \leq -1$, or to the current display format if $n=12$							
$z_1 \ n \rightarrow z_2$							
$z \ \text{'symp'} \rightarrow \text{'RND}(z, \text{symp})$							
$\text{'symp'} \ x \rightarrow \text{'RND}(\text{symp}, x)$							
$\text{'symp}_1 \ \text{'symp}_2 \rightarrow \text{'RND}(\text{symp}_1, \text{symp}_2)$							
$x_unit \ n \rightarrow x_unit$							
$x_unit \ \text{'symp'} \rightarrow \text{'RND}(x_unit, \text{symp})$							
$[\text{vector}_1] \ n \rightarrow [\text{vector}_2]$							
$[[\text{matrix}_1]] \rightarrow [[\text{matrix}_2]]$							
<i>Related Flag: -3</i>							
RNRM	(MTH)	MATR	NORM	RNRM	{ }		Command
Computes the maximum value of the sums of the absolute values of all elements over all rows							
$[\text{vector}] \rightarrow \text{row-norm}$							
$[[\text{matrix}]] \rightarrow \text{row-norm}$							
<i>Note: Since a vector is considered a 1-row matrix, RNRM returns the element in the vector having the largest absolute value.</i>							
ROLL	(\leftarrow)	(STACK)	ROLL				Command
Moves level $n+1$ object to level 1							
$\text{obj}_n \dots \text{obj}_1 \ n \rightarrow \text{obj}_{n-1} \dots \text{obj}_1 \ \text{obj}_n$							
ROLLD	(\leftarrow)	(STACK)	ROLLD				Command
Moves the level 2 object to level n							
$\text{obj}_1 \dots \text{obj}_n \ n \rightarrow \text{obj}_n \ \text{obj}_1 \dots \text{obj}_{n-1}$							
ROOT	(\leftarrow)	(SOLVE)	ROOT	ROOT			Command
Finds a numerical root							
$\text{'symp'} \ \text{'global'} \ \text{guess} \rightarrow \text{root}$							
$\text{'symp'} \ \text{'global'} \ \{ \text{guess}_1 \ \text{guess}_2 \} \rightarrow \text{root}$							
$\text{'symp'} \ \text{'global'} \ \{ \text{guess}_1 \ \text{guess}_2 \ \text{guess}_3 \} \rightarrow \text{root}$							
$\text{«program»} \ \text{'global'} \ \text{guess} \rightarrow \text{root}$							
$\text{«program»} \ \text{'global'} \ \{ \text{guess}_1 \ \text{guess}_2 \} \rightarrow \text{root}$							
$\text{«program»} \ \text{'global'} \ \{ \text{guess}_1 \ \text{guess}_2 \ \text{guess}_3 \} \rightarrow \text{root}$							

ROT	STACK ROT	Command
Moves the level 3 object to level 1 obj ₃ obj ₂ obj ₁ → obj ₂ obj ₁ obj ₃		
ROW+	MTH MATR ROW ROW+	G { } Command
Inserts a row vector into a matrix or a number into a vector [[matrix]] [vector] index → [[matrix]]' [vector] z index → [vector]'		
ROW-	MTH MATR ROW ROW-	G { } Command
Deletes a row from a matrix or a number from a vector [[matrix]] index → [[matrix]]' [deleted_row] [vector] index → [vector]' deleted_element		
ROW→	MTH MATR ROW ROW→	G Command
Transforms a series of row vectors into a matrix [vector ₁] ... [vector _n] n → [[matrix]]		
→ROW	MTH MATR ROW →ROW	G { } Command
Transforms a matrix into a series of row vectors [[matrix]] → [vector ₁] ... [vector _n] n		
RR	MTH BASE NXT BIT RR	{ } Command
Rotates right by one bit #n ₁ → #n ₂ <i>Related Flags:</i> -5 through -12		
RRB	MTH BASE NXT BYTE RRB	{ } Command
Rotates right by one byte #n ₁ → #n ₂ <i>Related Flags:</i> -5 through -12		
RREF	MTH MATR FACTR RREF	G Command
Computes the reduced row-echelon form of a rectangular matrix [[matrix]] → [[matrix]]' <i>Related Flag:</i> -54		
RRK	SOLVE DIFFE RRK	G Command
Computes the solution of the initial value problem for a differential equation using the Rosenbrock (3,4) and Runge-Kutta-Fehlberg (4,5) methods { t ₀ y ₀ f(t,y) } tolerance t _f → { t ₀ y ₀ f(t,y) } tolerance { t y f(t,y) ∂f/∂T ∂f/∂y } { tol hsize } t _f → { t y f(t,y) ∂f/∂T ∂f/∂y } tolerance		

RRKSTEP	 SOLVE	DIFFE	RRKS	G Command
Computes the next solution step of the initial value problem within a specified error tolerance using the Rosenbrock (3,4) Runge-Kutta-Fehlberg (4,5) methods				
{ t y f(t,y) $\partial f/\partial T$ $\partial f/\partial y$ } tol stepsize \rightarrow { t y f(t,y) $\partial f/\partial T$ $\partial f/\partial y$ } tol next-step				
RSBERR	 SOLVE	DIFFE	RSBER	G Command
Computes the change in solution and absolute error estimate using the Rosenbrock (3,4) and Runge-Kutta-Fehlberg (4,5) method				
{ t y f(t,y) $\partial f/\partial T$ $\partial f/\partial y$ } stepsize \rightarrow { t y f(t,y) $\partial f/\partial T$ $\partial f/\partial y$ } stepsize Δy error				
RSD	(MTH)	MATR	(NXT) RSD	{ } Command
Computes a correction to the solution of a system of equations				
[vector B] [[matrix A]] [vector Z] \rightarrow [vector B-AZ] [[matrix B]] [[matrix A]] [[matrix Z]] \rightarrow [[matrix B-AZ]]				
RSWP	(MTH)	MATR	ROW (NXT) RSWP	G Command
Row swap				
[[matrix]] index ₁ index ₂ \rightarrow [[matrix]]'				
R→B	(MTH)	BASE	R→B	{ } Command
Real-to-binary conversion				
n \rightarrow #n				
<i>Related Flags: -5 through -12</i>				
R→C	(MTH) (NXT)	CMPL	R→C	{ } Command
(PRG) TYPE (NXT) R→C				
Real-to-complex conversion				
x y \rightarrow (x,y) [R-array _{real}] [R-array _{imag}] \rightarrow [C-array]				
R→D	(MTH)	REAL	(NXT) (NXT) R→D	{ } Function
Radians-to-degrees conversion				
x \rightarrow (180/π)x 'symb' \rightarrow 'R→D(symb)'				
<i>Related Flag: -3</i>				
SAME	(PRG)	TEST	(NXT) SAME	Command
Tests two objects for equality				
obj ₁ obj ₂ \rightarrow T/F				
SBRK	 I/O (NXT)	SERIA	SBRK	Command
Sends serial break				
<i>Related Flag: -33</i>				











SCALE	 [PLOT] PPAR [NXT] SCALE	Command
Specifies x and y scale in units per 10 pixels x y →		
SCATRLOT	 [STAT] PLOT SCATR	Command
Draws a scatter plot of the data in ΣDAT		
SCATTER	 [PLOT] [NXT] STAT PTYPE SCATT	Command
Selects scatter plot		
SCHUR	[MTH] MATR FACTR SCHUR G { } Command	Command
Computes the Schur decomposition of a square matrix [[matrix A]] → [[matrix Q]] [[matrix T]]		
SCI	 [MODES] FMT SCI	Command
Sets Scientific display mode n →		
SCONJ	 [MEMORY] ARITH [NXT] SCON	Command
Conjugates the contents of a variable (see CONJ) 'name' →		
SDEV	 [STAT] IVAR SDEV	Command
Computes sample standard deviations of the data in ΣDAT → x → [x ₁ x ₂ ... x _n]		
SEND	 [I/O] SEND	Command
Sends object to another Kermit device 'local-name' → { { local-name remote-name } } → { local-name ₁ local-name ₂ ... } → { { local-name ₁ remote-name } local-name ₂ ... } → <i>Related Flags:</i> -33, -35, -39		
SEQ	[PRG] LIST PROC [NXT] SEQ G { } Command	Command
Generates list of results from repeated execution of object object 'name' start end step → { list }		
SERVER	   [I/O] SERVE SERVE	Command
Selects Kermit Server mode <i>Related Flags:</i> -33, -35, -36, -39		
SF	[PRG] TEST [NXT] [NXT] SF { } Command  [MODES] FLAG SF	Command
Sets a system or user flag ±n →		

SHOW	 SYMBOLIC	SHOW	Command
Resolves all name references or all name references except those in a list			
$\text{'symb}_1 \text{ 'name'} \rightarrow \text{'symb}_2$ $\text{'symb}_1 \{ \text{name}_1 \text{ name}_2 \dots \} \rightarrow \text{'symb}_2$			
<i>Related Flag:</i> -3			
SIDENS	 EQ LIB	UTILS SIDEN	G { } Function
Intrinsic density of silicon as a function of temperature. If no units are specified, T is assumed to be in K.			
$T \rightarrow \text{density (1/cm}^3\text{)}$ $\text{'symb'} \rightarrow \text{'SIDENS(symb')}$			
<i>Related Flag:</i> -3			
SIGN	MTH	REAL NXT SIGN	{ } } Function
	MTH NXT CMPL NXT SIGN		
Sign of a number. Complex numbers return a unit vector in the direction of z			
$x < 0 \rightarrow -1$ $x = 0 \rightarrow 0$ $x > 0 \rightarrow 1$ $z_1 \rightarrow z_2$ $x_unit \rightarrow y$ $\text{'symb'} \rightarrow \text{'SIGN(symb')}$			
<i>Related Flag:</i> -3			
SIN	 SIN		{ } ↓ ∂ } Function
Sine			
$z \rightarrow \sin z$ $\text{'symb'} \rightarrow \text{'SIN(symb')}$			
<i>Related Flags:</i> -3, -17, -18			
SINH	MTH	HYP SINH	{ } ↓ ∂ } Function
Hyperbolic sine			
$z \rightarrow \sinh z$ $\text{'symb'} \rightarrow \text{'SINH(symb')}$			
<i>Related Flag:</i> -3			
SINV	 MEMORY	ARITH NXT SINV	{ } Command
Inverts the contents of a variable			
$\text{'name'} \rightarrow$			

SIZE	[CHARS] SIZE	Command
	[PRG] LIST ELEM SIZE	
	[PRG] GROB [NXT] SIZE	
Finds the dimensions of an object		
	{ list } → objects	
	'algebraic' → objects	
	"string" → characters	
	[vector] → { elements }	
	[[matrix]] → { rows cols }	
	grob → #width #height	
	PICT → #width #height	
	unit_object → objects	
	other → 1	
SL	[MTH] BASE [NXT] BIT SL { }	Command
Shifts left by one bit		
	#n ₁ → #n ₂	
<i>Related Flags:</i> -5 through -12		
SLB	[MTH] BASE [NXT] BYTE SLB { }	Command
Shifts left by one byte		
	#n ₁ → #n ₂	
<i>Related Flags:</i> -5 through -12		
SLOPEFIELD	[PLOT] [NXT] 3D PTYPE	G Command
	SLOPE	
Selects slopefield plot		
SNEG	[MEMORY] ARITH [NXT] SNEG { }	Command
Negates the contents of a variable		
	'name' →	
SNRM	[MTH] MATR NORM SNRM	G { } Command
Computes the spectral norm of a matrix		
	[[matrix]] → spectral norm	
SOLVEQN	[EQ LIB] EQ LIB SOLVE	G Command
Places Equation Library equation(s) in solver, places corresponding picture in PICT if PICT_option is non-zero. For equation numbers, see <i>Equation Library Reference</i> .		
	subject_number title_number PICT_option →	
<i>Related Flags:</i> 60, 61		

Sort	[MTH] LIST SORT	G Command
	[PRG] LIST PROC [NXT] SORT	
Sorts the elements of a list in ascending order. If the list contains a series of lists they will be sorted using their first object as the key. <div>{ list } → { list' }</div>		
Sphere	[↵] [MODES] ANGL SPHER	G Command
	[MTH] VECTR [NXT] SPHER	
Sets polar/spherical mode		
SQ	[↵] [x²]	{ } ↓ ∂ Function
Squares a number or matrix <div> $z \rightarrow z^2$ [[matrix]] → [[matrix * matrix]] 'symb' → 'SQ(symb)' x_unit → x²_unit² </div>		
<i>Related Flag: -3</i>		
SR	[MTH] BASE [NXT] BIT SR	{ } Command
Shifts right by one bit <div>#n₁ → #n₂</div>		
<i>Related Flags: -5 through -12</i>		
SRAD	[MTH] MATR NORM SRAD	G { } Command
Computes the spectral radius of a square matrix <div>[[matrix A]] → spectral_radius</div>		
SRB	[MTH] BASE [NXT] BYTE SRB	{ } Command
Shifts right by one byte <div>#n₁ → #n₂</div>		
<i>Related Flags: -5 through -12</i>		
SRECV	[↵] [I/O] [NXT] SERIA SRECV	Command
Reads <i>n</i> characters from the I/O port, time-limited by the timeout specified by the command STIME. T/F is 1 for successful receive. <div>n → "string" T/F</div>		
<i>Related Flag: -33</i>		
START	[PRG] BRCH START START	Command
Begins START ... NEXT or START ... STEP <div>start end START loop-clause NEXT start end START loop-clause increment STEP</div>		
STD	[↵] [MODES] FMT STD	Command
Sets Standard display mode		





STEP	[PRG] BRCH START STEP [PRG] BRCH FOR STEP	Command
Ends FOR ... STEP or START ... STEP increment →		
STEQ	[↵] [PLOT] [↵] EQ [↵] [PLOT] [NXT] 3D [↵] EQ	Command
Stores into reserved variable <i>EQ</i> object →		
STIME	[↵] [I/O] [NXT] SERIA STIME	Command
Sets serial transmit/receive timeout. The valid range is 0 to 25.4 seconds. 0 means there is no time limit. The default timeout is 10 seconds. seconds →		
STO	[STO]	{ } Command
Stores an object into a variable obj 'name' → obj :port:name → obj name(position) → grob <i>PICT</i> → backup port → library port →		
STOALARM	[↵] [TIME] ALARM STOAL	Command
Stores alarm in system alarm list. Repeat interval is specified in <i>ticks</i> (8192 per second). time → alarm_number { date } → alarm_number { date time } → alarm_number { date time action } → alarm_number { date time action repeat } → alarm_number <i>Related Flags:</i> -42, -43, -44		
STOF	[↵] [MODES] FLAG [NXT] STOF	Command
Sets the system flags or the system and user flags according to the value of two binary integers in a list #system → { #system #user } → <i>Related Flags:</i> -5 through -10 <i>Note:</i> The wordsize should be set to 64 bits		

STOKEYS	 MODES	KEYS	STOK	Command
Makes multiple user-key assignments. Including S activates standard key definitions. Including SKEY restores standard key assignment.				
S →				
{ SKEY rc.p } →				
{ obj ₁ rc.p ₁ ... obj _n rc.p _n } →				
{ S obj ₁ rc.p ₁ ... obj _n rc.p _n } →				
Related Flags: -61, -62				
STO+	 MEMORY	ARITH	STO+	{ } Command
Storage addition (see +)				
object 'name' →				
'name' object →				
STO-	 MEMORY	ARITH	STO-	{ } Command
Storage subtraction (see -)				
object 'name' →				
'name' object →				
STO*	 MEMORY	ARITH	STO*	{ } Command
Storage multiplication (see *)				
object 'name' →				
'name' object →				
STO/	 MEMORY	ARITH	STO/	{ } Command
Storage division (see /)				
object 'name' →				
'name' object →				
STOΣ	 STAT	DATA	 STOΣ	Command
Stores into reserved variable ΣDAT				
object →				
STR→				Command
Evaluates the commands defined by a string after removing the “ ” delimiters				
“string” →				
→STR	 CHARS	 NXT	→STR	Command
 PRG TYPE →STR				
Converts an object to a string				
object → “string”				
Related Flags: -5 through -12, -49, -50				

STREAM	(PRG) LIST PROC STREA	G Command
Executes <i>object</i> using first two objects in <i>list</i> , then executes <i>object</i> using third object in <i>list</i> and the previous result until <i>list</i> is exhausted { list } object → object		
STWS	(MTH) BASE (NXT) STWS	Command
Sets the binary integer wordsize n → #n → <i>Related Flags:</i> -5 through -12		
SUB	(↵) (CHARS) SUB (PRG) LIST ELEM (NXT) SUB (PRG) GROB SUB	Command
Extracts a portion of a list, string, or grob { list } start end → { sublist } “string” start end → “substring” grob (x ₁ ,y ₁) (x ₂ ,y ₂) → subgrob grob { #x ₁ #y ₁ } { #x ₂ #y ₂ } → subgrob <i>PICT</i> (x ₁ ,y ₁) (x ₂ ,y ₂) → subgrob <i>PICT</i> { #x ₁ #y ₁ } { #x ₂ #y ₂ } → subgrob		
SVD	(MTH) MATR FACTR (SVD)	G { } Command
Computes the singular value decomposition of a matrix [[matrix A]] → [[matrix U]] [[matrix V]] [vector S]		
SVL	(MTH) MATR FACTR (NXT) SVL	G { } Command
Computes the singular values of a matrix [[matrix A]] → [vector S]		
SWAP	(↵) (SWAP)	Command
Swaps the objects in levels 1 and 2 obj ₂ obj ₁ → obj ₁ obj ₂		
SYSEVAL		Command
Executes a system object at the specified address #n →		
→TAG	(PRG) TYPE →TAG	{ } Command
Tags an object with another object obj “tag” → :tag:obj obj ‘name’ → :name:obj obj x → :x:obj		

TAIL	[↵] [CHARS] [NXT] TAIL	G Command
	[PRG] LIST ELEM [NXT] TAIL	
Returns a list less its first object or a string less its first character $\{ \text{obj}_1 \text{ obj}_2 \dots \text{obj}_n \} \rightarrow \{ \text{obj}_2 \dots \text{obj}_n \}$ "ABCDE" \rightarrow "BCDE"		
TAN	[TAN]	{ } ↓ ∂ / Function
Tangent $z \rightarrow \tan z$ 'symb' \rightarrow 'TAN(symb)' <i>Related Flags:</i> -3, -17, -18, -22		
TANH	[MTH] HYP TANH	{ } ↓ ∂ / Function
Hyperbolic tangent $z \rightarrow \tanh z$ 'symb' \rightarrow 'TANH(symb)' <i>Related Flag:</i> -3		
TAYLR	[↵] [SYMBOLIC] TAYLR	Command
Computes a Taylor series approximation 'symb ₁ ' 'global' degree \rightarrow 'symb ₂ '		
TDELTA	[↵] [EQ LIB] UTILS [NXT] TDELT	G Function
Calculates temperature difference $T_1 \ T_2 \rightarrow \text{difference}$ 'symb' x \rightarrow 'TDELTA(symb,x)' x 'symb' \rightarrow 'TDELTA(x,symb)' 'symb ₁ ' 'symb ₂ ' \rightarrow 'TDELTA(symb ₁ ,symb ₂)' <i>Related Flag:</i> -3 <i>Note:</i> Values returned by TDELTA have level 2 units		
TEACH		G Command
Creates examples directory in the VAR menu		
TEXT	[PRG] [NXT] OUT TEXT	Command
Selects the stack display		
THEN	[PRG] BRCH IF THEN [PRG] BRCH CASE THEN [PRG] [NXT] ERROR IFERR THEN	Command
Begins true-clause of IF, CASE, or IFERR structures T/F \rightarrow		




TICKS	TIME TICKS	Command
Returns time in binary integer clock ticks (8192 per second) → #n		
TIME	TIME TIME	Command
Returns current time as number in 24-hour format → HH.MMSSs		
→TIME	TIME +TIM	Command
Sets the system time using 24-hour format HH.MMSSs →		
TINC	EQ LIB UTILS NXT TINC	G Function
Adds temperature increment T_1 increment → T_2 'symb' x → 'TINC(symb,x)' x 'symb' → 'TINC(x,symb)' 'symb ₁ ' 'symb ₂ ' → 'TINC(symb ₁ ,symb ₂)'		
<i>Related Flag:</i> -3 <i>Note:</i> Values returned by TINC have level 2 units		
TLINE	PRG PICT TLINE	Command
Toggles pixels on a straight line $(x_1, y_1) (x_2, y_2)$ → { #x ₁ #y ₁ } { #x ₂ #y ₂ } →		
TMENU	MODES MENU TMEN	Command
Displays temporary built-in or list-defined menu (see <i>Built-in Menus</i>) mm.pp → 'list-name' → { names and commands } → <i>Note:</i> TMENU does not affect the contents of the variable CST		
TOT	STAT 1VAR TOT	Command
Sums the columns in Σ DAT → x → [x ₁ x ₂ ... x _m]		
TRACE	MTH MATR NORM NXT TRACE G { }	Command
Computes the trace (sum of diagonal elements) of a square matrix [[matrix]] → trace		










































TRANSIO  I/O IOPAR TRAN	Command
Selects character translation mode for I/O	
$\begin{array}{ll} n & \rightarrow \\ 0 & \text{No translation} \\ 1 & \text{CR to CR/LF (default)} \\ 2 & \text{Chars 128–159} \\ 3 & \text{Chars 128–255} \end{array}$	
TRN (MTH) MATR MAKE TRN	{ } Command
Transposes a matrix	
$\begin{array}{ll} [[\text{matrix}_1]] & \rightarrow [[\text{matrix}_2]] \\ \text{'name'} & \rightarrow \end{array}$	
TRNC (MTH) REAL (NXT) (NXT) TRNC	{ } Function
Truncates a number to n decimal places if $0 \leq n \leq 11$, to n significant digits if $-11 \leq n \leq -1$, or to the current display format if $n=12$	
$\begin{array}{ll} z_1 \ n & \rightarrow z_2 \\ z \ \text{'symb'} & \rightarrow \text{'TRNC}(z,\text{symb})' \\ \text{'symb'} \ x & \rightarrow \text{'TRNC}(\text{symb},x)' \\ \text{'symb}_1 \ \text{'symb}_2 & \rightarrow \text{'TRNC}(\text{symb}_1,\text{symb}_2) \\ x_unit \ n & \rightarrow x_unit \\ x_unit \ \text{'symb'} & \rightarrow \text{'TRNC}(x_unit,\text{symb})' \\ [\text{vector}_1] \ n & \rightarrow [\text{vector}_2] \\ [[\text{matrix}_1]] & \rightarrow [[\text{matrix}_2]] \end{array}$	
<i>Related Flag: -3</i>	
TRUTH  (PLOT) PTYPE TRUTH	Command
Selects truth plot	
TSTR  (TIME) (NXT) (NXT) TSTR	{ } Command
Converts date and time numbers to string form	
$\text{date time} \rightarrow \text{"string"}$	
<i>Related Flags: -41, -42</i>	
TVARS  (MEMORY) DIR TVARS	{ } Command
Lists the variables of specified type found in the current directory (see <i>Object Types</i>)	
$\text{type} \rightarrow \{ \text{names} \}$	
TVM	G Command
Displays the TVM menu	
TVMBEG	G Command
Sets Begin payment mode	






TVMEND	G Command
Sets End payment mode	
TVMROOT [SOLVE] TVM TVMR	G Command
Solves for TVM variable using the other TVM variables ‘name’ → value	
<i>Related Flag:</i> -14	
TYPE TEST TYPE	Command
TYPE TYPE	
Returns the type of an object (see <i>Object Types</i>) object → type	
UBASE [UNITS] UBASE	{ } Function
Converts unit object to SI base units x → x x_units → y_base-units ‘symb’ → ‘UBASE(symb)’	
<i>Related Flag:</i> -3	
UFACT [UNITS] UFACT	{ } Command
Factors specified compound unit x y_units → x x_units ₁ y_units ₂ → x’_units ₂ * units ₃	
→UNIT TYPE →UNIT	{ } Command
[UNITS] →UNIT	
Combines a number and unit object to create a new unit object x y_units → x_units	
UNTIL BRCH DO UNTIL	Command
Begins test-clause of DO ... UNTIL ... END	
UPDIR [UP]	Command
Makes parent directory the current directory	
UTPC PROB UTPC	{ } Command
Upper-tail Chi-Square distribution d.o.f. x → utpc(d.o.f.,x)	
UTPF PROB UTPF	{ } Command
Upper-tail F-distribution d.o.f. ₁ d.o.f. ₂ x → utpf(d.o.f. ₁ , d.o.f. ₂ , x)	
UTPN PROB UTPN	{ } Command
Upper-tail normal distribution mean variance x → utpn(mean, variance, x)	























UTPT	(MTH)	(NXT)	PROB	(NXT)	UTPT	{ }	Command		
Upper-tail Student's t-distribution									
				d.o.f.	x	→	utpt(d.o.f.,x)		
UVAL	(↶)	(UNITS)	UVAL					{ }	Function
Returns scalar portion of unit object									
				x	→	x			
				x_unit	→	x			
				'symb'	→	'UVAL(symb)'			
<i>Related Flag: -3</i>									
→V2	(MTH)	VECTR	→V2					{ }	Command
Combines two real numbers into 2D vector or complex number									
				x y	→	[x y]			
				x y	→	[x ∠ y]			
				x y	→	(x,y)			
				x y	→	(x,∠y)			
<i>Related Flags: -15, -16, -19</i>									
→V3	(MTH)	VECTR	→V3					{ }	Command
Combines three real numbers into 3D vector									
				x y z	→	[x y z]			
				x y z	→	[x ∠ y ∠ z]			
				x y z	→	[x ∠ y ∠ z ∠ 0]			
<i>Related Flags: -15, -16</i>									
V→	(MTH)	VECTR	V→					{ }	Command
Separates a 2- or 3-element vector. If there are more than 3 elements, the current Coordinate System (flags -15 and -16) is ignored.									
				[x y]	→	x y			
				[x _r ∠ y _θ]	→	x _r y _θ			
				[x y z]	→	x y z			
				[x _r ∠ y _θ z]	→	x _r y _θ z			
				[x _r ∠ y _θ ∠ z ∠ 0]	→	x _r y _θ z ∠ 0			
				(x,y)	→	x y			
				(x _r ,∠y)	→	x _r ∠ y _θ			
				[x ₁ x ₂ ... x _n]	→	x ₁ x ₂ ... x _n			
<i>Related Flags: -15, -16</i>									

VAR	[STAT] IVAR [NXT] VAR	Command
Sample variances of Σ DATA data in columns specified by COL Σ		
\rightarrow x		
\rightarrow [x ₁ x ₂ ... x _m]		
VARS	[MEMORY] DIR VARS	Command
Returns list of variables in the current directory		
\rightarrow { names }		
VERSION		G Command
Returns strings identifying the operating system version & copyright		
\rightarrow "version" "copyright"		
VTYPE	[PRG] TYPE [NXT] [NXT] VTYPE { }	Command
Returns the type of an object in the named variable, or -1 if the variable is nonexistent (see <i>Object Types</i>)		
'name' \rightarrow type		
:port:name \rightarrow type		
*W	[PLOT] PPAR [NXT] *W	Command
Multiplies the horizontal plot scale by specified factor (alters PPAR)		
x \rightarrow		
WAIT	[PRG] [NXT] IN WAIT	Command
Pauses program execution or waits for a key		
seconds \rightarrow		
0 \rightarrow rc.p <i>Doesn't update menu</i>		
-1 \rightarrow rc.p <i>Displays current menu</i>		
WHILE	[PRG] BRCH WHILE WHILE	Command
Begins WHILE ... REPEAT .. END		
WHILE test-clause REPEAT loop-clause END		
WIREFRAME	[PLOT] [NXT] 3D P TYPE WIREF	G Command
Selects wireframe plot		
WSLOG		Command
Returns four strings indicating the time, date, and source of the four most recent system halts (see <i>System Operations</i>)		
\rightarrow "string ₄ " "string ₃ " "string ₂ " "string ₁ "		
<i>Related Flag:</i> -42		
XCOL	[STAT] ΣPAR XCOL	Command
Specifies Σ DATA column as the independent variable		
x-column \rightarrow		

XMIT	 I/O NXT SERIAL XMIT	Command
Sends string through I/O port without Kermit		
“string” → 1 “string” → “unsent string” 0		
<i>Related Flag:</i> -33		
XOR	MTH BASE NXT LOGIC XOR { } Function PRG TEST NXT XOR	
Logical or binary XOR		
#n ₁ #n ₂ → #n ₃ x y → T/F x ‘symb’ → ‘x XOR symb’ ‘symb’ x → ‘symb XOR x’ ‘symb ₁ ’ ‘symb ₂ ’ → ‘symb ₁ XOR symb ₂ ’ “string ₁ ” “string ₂ ” → “string ₃ ”		
<i>Related Flags:</i> -3, -5 through -10		
<i>Note:</i> String arguments must have the same length.		
XPON	MTH REAL NXT XPON { } Function	
Returns the exponent of a number		
x → n ‘symb’ → ‘XPON(symb)’		
<i>Related Flag:</i> -3		
XRECV	 I/O NXT XRECV	G Command
Receive an object using Xmodem protocol (binary mode only)		
‘name’ →		
<i>Related Flag:</i> -33		
XRNG	 PLOT PPAR XRNG	Command
Specifies x-axis plotting range		
x _{min} x _{max} →		

XROOT		{ } Function
Returns x^{th} root of y		
$y \ x \rightarrow \sqrt[x]{y}$		
$y_unit \ x \rightarrow \sqrt[x]{y_unit}^{1/x}$		
$y \ 'symb' \rightarrow 'XROOT(symb,y)'$		
$'symb' \ x \rightarrow 'XROOT(x,symb)'$		
$'symb_1' \ 'symb_2' \rightarrow 'XROOT(symb_2,symb_1)'$		
$y_unit \ 'symb' \rightarrow 'XROOT(symb,y_unit)'$		
<i>Related Flag: -3</i>		
XSEND	     XSEN	G Command
Sends an object using Xmodem protocol (binary mode only)		
$'name' \rightarrow$		
<i>Related Flag: -33</i>		
XVOL	   3D VPAR XVOL	G Command
Sets the width of the 3D plotting volume		
$x_{\text{left}} \ x_{\text{right}} \rightarrow$		
XXRNG	   3D VPAR XXRN	G Command
Sets the width of 3D target mapping range for gridmap and parsurface plots		
$x_{\text{min}} \ x_{\text{max}} \rightarrow$		
YCOL	          YCOL	Command
Specifies a Σ DAT column as the dependent variable		
$y\text{-column} \rightarrow$		
YRNG	          YRNG	Command
Specifies y-axis plotting range		
$y_{\text{min}} \ y_{\text{max}} \rightarrow$		
YSLICE	   3D PTYPE YSLIC	G Command
Selects yslice plot		
YVOL	   3D VPAR YVOL	G Command
Sets the depth of the 3D plotting volume		
$y_{\text{near}} \ y_{\text{far}} \rightarrow$		
YYRNG	   3D VPAR YYRN	G Command
Sets depth of 3D target mapping range for gridmap and parsurface plots		
$y_{\text{min}} \ y_{\text{max}} \rightarrow$		

ZFACTOR  EQ LIB UTILS ZFACT G { } Function Calculates gas compressibility factor Z Tr Pr → Z 'symb' x → 'ZFACTOR(symb,x)' x 'symb' → 'ZFACTOR(x,symb)' 'symb ₁ ' 'symb ₂ ' → 'ZFACTOR(symb ₁ ,symb ₂)' <i>Related Flag:</i> -3
ZVOL  PLOT NXT 3D VPAR ZVOL G Command Sets the height of the 3D plotting volume Z _{low} Z _{high} →
$\sqrt{\quad}$  { } ↓ ∂ } Function Square root z → \sqrt{z} x _{unit} → $\sqrt{x_{unit}}^{1/2}$ 'symb' → ' $\sqrt{(symb)}$ ' <i>Related Flags:</i> -1, -3
\int  { } ∂ } Function Integral lower_limit upper_limit 'integrand' 'name' → integral '(lower_limit, upper_limit, integrand, name)' <i>Related Flags:</i> -3, -45 through -50 <i>Notes:</i> 1) <i>name</i> is the variable of integration. 2) Set Numerical Results mode (flag -3) to perform a numerical integration on the stack. 3) The display mode (such as 2 FIX) specifies the accuracy factor for numerical integration, and the uncertainty of integration is stored in reserved variable <i>IERR</i>
∂  { } ∂ } Function Derivative 'symb ₁ ' 'name' → 'symb ₂ ' <i>Complete</i> '∂name(expression)' <i>Stepwise</i> <i>Related Flag:</i> -3 <i>Note:</i> <i>name</i> is the variable of differentiation

π	 	∂ Function
Symbolic constant π		
	$\rightarrow 3.14159265359$	
	$\rightarrow '\pi'$	
<i>Related Flags: -2, -3</i>		
Σ	 	{ } ∂ Function
Summation		
	'summation_index' initial_value final_value 'summand' \rightarrow sum	
	' $\Sigma(\text{summation_index}=\text{initial_value},\text{final_value},\text{summand})$ '	
<i>Related Flag: -3</i>		
ΣX	  SUMS ΣX	Command
Sum of data in independent ΣDAT column		
	$\rightarrow \Sigma X_i$	
ΣX^2	  SUMS ΣX^2	Command
Sum of squares of data in independent ΣDAT column		
	$\rightarrow \Sigma X_i^2$	
ΣY	  SUMS ΣY	Command
Sum of data in dependent ΣDAT column		
	$\rightarrow \Sigma Y_i$	
ΣY^2	  SUMS ΣY^2	Command
Sum of squares of data in dependent ΣDAT column		
	$\rightarrow \Sigma Y_i^2$	
$\Sigma X*Y$	  SUMS $\Sigma X*Y$	Command
Sum of products of data in independent and dependent ΣDAT columns		
	$\rightarrow \Sigma X_i Y_i$	
$\Sigma +$	  DATA $\Sigma +$   STAT DATA $\Sigma +$	Command
Appends one or more data points to ΣDAT		
	x \rightarrow	
	[vector] \rightarrow	
	[[matrix]] \rightarrow	
$\Sigma -$	  DATA $\Sigma -$   STAT DATA $\Sigma -$	Command
Deletes last row from ΣDAT		
	$\rightarrow x$	
	$\rightarrow [\text{vector}]$	

< **[PRG]** TEST < { } Function

Less-than comparison

$x \ y \rightarrow x < y \text{ (T/F)}$
 $\text{"string}_1 \ \text{"string}_2 \rightarrow \text{T/F}$
 $x_unit_1 \ y_unit_2 \rightarrow \text{T/F}$
 $x \ \text{'symb'} \rightarrow \text{'x' < symb'}$
 $\text{'symb'} \ x \rightarrow \text{'symb' < x'}$
 $x_unit \ \text{'symb'} \rightarrow \text{'x_unit' < symb'}$
 $\text{'symb'} \ x_unit \rightarrow \text{'symb' < x_unit'}$
 $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'symb}_1 < \text{'symb}_2'$
 $\text{:tag:object object} \rightarrow \text{T/F}$
 $\text{object :tag:object} \rightarrow \text{T/F}$
 $\text{object object} \rightarrow \text{T/F}$

Related Flag: -3

Notes:

- 1) Strings are compared alphabetically
- 2) Units must be dimensionally consistent
- 3) Tags are dropped before the comparison

> **[PRG]** TEST > { } Function

Greater-than comparison

$x \ y \rightarrow x > y \text{ (T/F)}$
 $x_unit_1 \ y_unit_2 \rightarrow \text{T/F}$
 $x \ \text{'symb'} \rightarrow \text{'x' > symb'}$
 $\text{'symb'} \ x \rightarrow \text{'symb' > x'}$
 $x_unit \ \text{'symb'} \rightarrow \text{'x_unit' > symb'}$
 $\text{'symb'} \ x_unit \rightarrow \text{'symb' > x_unit'}$
 $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'symb}_1 > \text{'symb}_2'$
 $\text{:tag:object object} \rightarrow \text{T/F}$
 $\text{object :tag:object} \rightarrow \text{T/F}$
 $\text{object object} \rightarrow \text{T/F}$

Related Flag: -3

Notes:

- 1) Strings are compared alphabetically
- 2) Units must be dimensionally consistent
- 3) Tags are dropped before the comparison

\leq **[PRG]** TEST \leq { } Function

Less-than-or-equal comparison

$x \ y \rightarrow x \leq y \text{ (T/F)}$
 $\text{"string}_1 \text{" "string}_2 \rightarrow \text{T/F}$
 $x_unit_1 \ y_unit_2 \rightarrow \text{T/F}$
 $x \ \text{'symb'} \rightarrow \text{'x} \leq \text{symb'}$
 $\text{'symb'} \ x \rightarrow \text{'symb} \leq x$
 $x_unit \ \text{'symb'} \rightarrow \text{'x_unit} \leq \text{symb'}$
 $\text{'symb'} \ x_unit \rightarrow \text{'symb} \leq x_unit$
 $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'symb}_1 \leq \text{symb}_2$
 $\text{:tag:object} \ \text{object} \rightarrow \text{T/F}$
 $\text{object} \ \text{:tag:object} \rightarrow \text{T/F}$
 $\text{object} \ \text{object} \rightarrow \text{T/F}$

Related Flag: -3

Notes:

- 1) Strings are compared alphabetically
- 2) Units must be dimensionally consistent
- 3) Tags are dropped before the comparison

\geq **[PRG]** TEST \geq { } Function

Greater-than-or-equal comparison

$x \ y \rightarrow x \geq y \text{ (T/F)}$
 $x_unit_1 \ y_unit_2 \rightarrow \text{T/F}$
 $x \ \text{'symb'} \rightarrow \text{'x} \geq \text{symb'}$
 $\text{'symb'} \ x \rightarrow \text{'symb} \geq x$
 $x_unit \ \text{'symb'} \rightarrow \text{'x_unit} \geq \text{symb'}$
 $\text{'symb'} \ x_unit \rightarrow \text{'symb} \geq x_unit$
 $\text{'symb}_1 \ \text{'symb}_2 \rightarrow \text{'symb}_1 \geq \text{symb}_2$
 $\text{:tag:object} \ \text{object} \rightarrow \text{T/F}$
 $\text{object} \ \text{:tag:object} \rightarrow \text{T/F}$
 $\text{object} \ \text{object} \rightarrow \text{T/F}$

Related Flag: -3

Notes:

- 1) Strings are compared alphabetically
- 2) Units must be dimensionally consistent
- 3) Tags are dropped before the comparison

≠ **PRG** **TEST** **≠** **{ }** Function
 Not-equal comparison

$x \ y \rightarrow x \neq y \text{ (T/F)}$
 $x \ z \rightarrow \text{T/F}$
 $z \ x \rightarrow \text{T/F}$
 $x_unit_1 \ y_unit_2 \rightarrow \text{T/F}$
 $x \ 'symb' \rightarrow 'x \neq symb'$
 $'symb' \ x \rightarrow 'symb \neq x'$
 $x_unit \ 'symb' \rightarrow 'x_unit \neq symb'$
 $'symb' \ x_unit \rightarrow 'symb \neq x_unit'$
 $'symb_1' \ 'symb_2' \rightarrow 'symb_1 \neq symb_2'$
 $:tag:object \ object \rightarrow \text{T/F}$
 $object \ :tag:object \rightarrow \text{T/F}$
 $object \ object \rightarrow \text{T/F}$

Related Flag: -3

Notes:

- 1) Strings are compared alphabetically
- 2) Units must be dimensionally consistent
- 3) Real-complex comparisons assume the imaginary part is 0
- 4) Tags are dropped before the comparison

= **EQ** **{ }** Function
 Equal operator. Creates an equation from two arguments

$z_1 \ z_2 \rightarrow 'z_1 = z_2'$
 $z \ 'symb' \rightarrow 'z = symb'$
 $'symb' \ z \rightarrow 'symb = z'$
 $x \ y_unit \rightarrow 'x = y_unit'$
 $x_unit \ y \rightarrow 'x_unit = y'$
 $x_unit \ y_unit \rightarrow 'x_unit = y_unit'$
 $'symb' \ x_unit \rightarrow 'symb = x_unit'$
 $x_unit \ 'symb' \rightarrow 'x_unit = symb'$
 $'symb_1' \ 'symb_2' \rightarrow 'symb_1 = symb_2'$

Related Flag: -3

== [PRG] TEST ==	Function
Logical equality comparison <div style="text-align: center;"> $x \ y \rightarrow x==y \text{ (T/F)}$ $x \ z \rightarrow \text{T/F}$ $z \ x \rightarrow \text{T/F}$ $x_unit_1 \ y_unit_2 \rightarrow \text{T/F}$ $x \ 'symb' \rightarrow 'x==symb'$ $'symb' \ x \rightarrow 'symb==x'$ $x_unit \ 'symb' \rightarrow 'x_unit==symb'$ $'symb' \ x_unit \rightarrow 'symb==x_unit'$ $'symb_1' \ 'symb_2' \rightarrow 'symb_1==symb_2'$ $:tag:object \ object \rightarrow \text{T/F}$ $object \ :tag:object \rightarrow \text{T/F}$ $object \ object \rightarrow \text{T/F}$ </div> <p><i>Related Flag:</i> -3</p> <p><i>Notes:</i></p> <ol style="list-style-type: none"> 1) Units must be dimensionally consistent 2) Real-complex comparisons assume the imaginary part is 0 3) Tags are dropped before the comparison 	
→ [P] [Q]	Command
Assigns local variable(s) <div style="text-align: center;"> $obj_1 \dots obj_n \rightarrow$ </div>	
 [↩] [SYMBOLIC] [NXT] 	∂ Function
WHERE function. Substitutes symbolics for names in a symbolic expression. <div style="text-align: center;"> $'symb_{old}' \{ name_1 \ symb_1 \dots name_n \ symb_n \} \rightarrow 'symb_{new}'$ $z \ 'symb_{old}' \{ name_1 \ symb_1 \dots name_n \ symb_n \} \rightarrow z$ $'symb_{old}'(name_1=symb_1, \dots, name_n=symb_n)'$ </div> <p><i>Related Flag:</i> -3</p>	

+



↓ ∂ Function




Adds two objects

z_1	z_2	→	z_1+z_2
#n	m	→	#n+m
n	#m	→	#n+m
#n	#m	→	#n+m
x_unit	y_unit	→	x+y_unit
'sybm ₁ '	'sybm ₂ '	→	'sybm ₁ +sybm ₂ '
z	'sybm'	→	'z+sybm'
'sybm'	z	→	'sybm+z'
'sybm'	x_unit	→	'sybm+x_unit'
x_unit	'sybm'	→	'x_unit+sybm'
[vector ₁]	[vector ₂]	→	[vector ₁ +vector ₂]
[[matrix ₁]]	[[matrix ₂]]	→	[[matrix ₁ +matrix ₂]]
grob ₁	grob ₂	→	grob ₃
"abc"	"def"	→	"abcdef"
"string"	object	→	"stringobject"
object	"string"	→	"objectstring"
{ list }	object	→	{ list object }
object	{ list }	→	{ object list }
{ obj ₁ obj ₂ }	{ obj ₃ obj ₄ }	→	{ obj ₁ obj ₂ obj ₃ obj ₄ }

Related Flags: -3, -5 through -10*Notes:*

- 1) Grobs must have the same dimensions
- 2) →STR is executed on objects added to strings
- 3) Units must be dimensionally consistent
- 4) For element-wise addition of objects in lists, use ADD

<p>—  Subtracts two objects</p>	<p>{ } ↓ ∂ Function</p>
	$ \begin{aligned} z_1 \ z_2 &\rightarrow z_1 - z_2 \\ \#n \ m &\rightarrow \#n - m \\ n \ \#m &\rightarrow \#n - m \\ \#n \ \#m &\rightarrow \#n - m \\ x_unit \ y_unit &\rightarrow x - y_unit \\ z \ 'symb' &\rightarrow 'z - symb' \\ 'symb' \ z &\rightarrow 'symb - z' \\ 'symb_1' \ 'symb_2' &\rightarrow 'symb_1 - symb_2' \\ 'symb' \ x_unit &\rightarrow 'symb - x_unit' \\ x_unit \ 'symb' &\rightarrow 'x_unit - symb' \\ [vector_1] \ [vector_2] &\rightarrow [vector_1 - vector_2] \\ [[matrix_1]] \ [[matrix_2]] &\rightarrow [[matrix_1 - matrix_2]] \end{aligned} $ <p><i>Related Flags:</i> -3, -5 through -10</p> <p><i>Note:</i> Units must be dimensionally consistent</p>
<p>*  Multiplies two objects</p>	<p>{ } ↓ ∂ Function</p>
	$ \begin{aligned} z_1 \ z_2 &\rightarrow z_1 * z_2 \\ \#n \ m &\rightarrow \#n * m \\ n \ \#m &\rightarrow \#n * m \\ \#n \ \#m &\rightarrow \#n * m \\ [vector] \ z &\rightarrow [vector * z] \\ z \ [vector] &\rightarrow [vector * z] \\ [[matrix]] \ [vector] &\rightarrow [matrix * vector] \\ [[matrix_1]] \ [[matrix_2]] &\rightarrow [[matrix * vector]] \\ z \ 'symb' &\rightarrow 'z * symb' \\ 'symb' \ z &\rightarrow 'symb * z' \\ 'symb_1' \ 'symb_2' &\rightarrow 'symb_1 * symb_2' \\ x \ y_unit &\rightarrow x * y_unit \\ x_unit \ y &\rightarrow x * y_unit \\ x_unit_1 \ y_unit_2 &\rightarrow x * y_unit_1 * unit_2 \\ x_unit \ 'symb' &\rightarrow '(x_unit) * (symb)' \\ 'symb' \ x_unit &\rightarrow '(symb) * (x_unit)' \end{aligned} $ <p><i>Related Flags:</i> -3, -5 through -10</p>

/		{ } ↓ ∂ Function
 SOLVE	SYS	/
Divides two objects		
	$z_1 \ z_2$	$\rightarrow z_1/z_2$
	$\#n \ m$	$\rightarrow \#n/m$
	$n \ \#m$	$\rightarrow \#n/m$
	$\#n \ \#m$	$\rightarrow \#n/m$
	[vector] z	\rightarrow [vector/ z]
	[vector] [[matrix]]	\rightarrow [[vector/matrix]]
	z 'symb'	\rightarrow 'z/symb'
	'symb' z	\rightarrow 'symb/ z '
	'symb ₁ ' 'symb ₂ '	\rightarrow '(symb ₁)/(symb ₂)'
	x y_{unit}	$\rightarrow x/y_{\text{unit}}$
	x_{unit} y	$\rightarrow x/y_{\text{unit}}$
	x_{unit_1} y_{unit_2}	$\rightarrow x/y_{\text{unit}_1/\text{unit}_2}$
	x_{unit} 'symb'	\rightarrow '(x_{unit})/(symb)'
	'symb' x_{unit}	\rightarrow '(symb)/(x_{unit})'
Related Flags: -3, -5 through -10		
^		{ } ↓ ∂ Function
Raises a number to a power		
	$z_1 \ z_2$	$\rightarrow z_1^{z_2}$
	z 'symb'	\rightarrow 'z^(symb)'
	'symb' z	\rightarrow '(symb)^z'
	'symb ₁ ' 'symb ₂ '	\rightarrow '(symb ₁)^(symb ₂)'
	x_{unit} y	$\rightarrow x^{y_{\text{unit}}}$
	x_{unit} 'symb'	\rightarrow '(x_{unit})^(symb)'
	'symb' x_{unit}	\rightarrow '(symb)^(x_{unit})'
Related Flag: -3		
!	(MTH) (NXT) PROB	! { } Function
Factorial or gamma function		
	n	$\rightarrow n!$
	x	$\rightarrow \Gamma(x+1)$
	'symb'	\rightarrow '(symb)!'
Related Flags: -3, -20, -21		

% (MTH) REAL (NXT) % { } Function
Percent

$x \ y \rightarrow xy/100$
 $x \text{ 'symb' } \rightarrow \text{'%(x,symb)'}$
 $\text{'symb' } x \rightarrow \text{'%(symb,x)'}$
 $\text{'symb}_1 \text{ 'symb}_2 \rightarrow \text{'%(symb}_1,\text{symb}_2)'$
 $x_unit \ y \rightarrow xy/100_unit$
 $x_unit \text{ 'symb' } \rightarrow \text{'%(x_unit,symb)'}$
 $\text{'symb' } x_unit \rightarrow \text{'%(symb,x_unit)'}$
 $x \ y_unit \rightarrow xy/100_unit$

Related Flag: -3

%CH (MTH) REAL (NXT) %CH { } Function

Percent change from x to y as a percentage of x

$x \ y \rightarrow 100(y-x)/x$
 $x \text{ 'symb' } \rightarrow \text{'%CH(x,symb)'}$
 $\text{'symb' } x \rightarrow \text{'%CH(symb,x)'}$
 $\text{'symb}_1 \text{ 'symb}_2 \rightarrow \text{'%CH(symb}_1,\text{symb}_2)'$
 $x_unit \ y_unit \rightarrow 100(y-x)/x$
 $x_unit \text{ 'symb' } \rightarrow \text{'%CH(x_unit,symb)'}$
 $\text{'symb' } x_unit \rightarrow \text{'%CH(symb,x_unit)'}$

Related Flag: -3

Note: Units must be dimensionally consistent

%T (MTH) REAL (NXT) %T { } Function

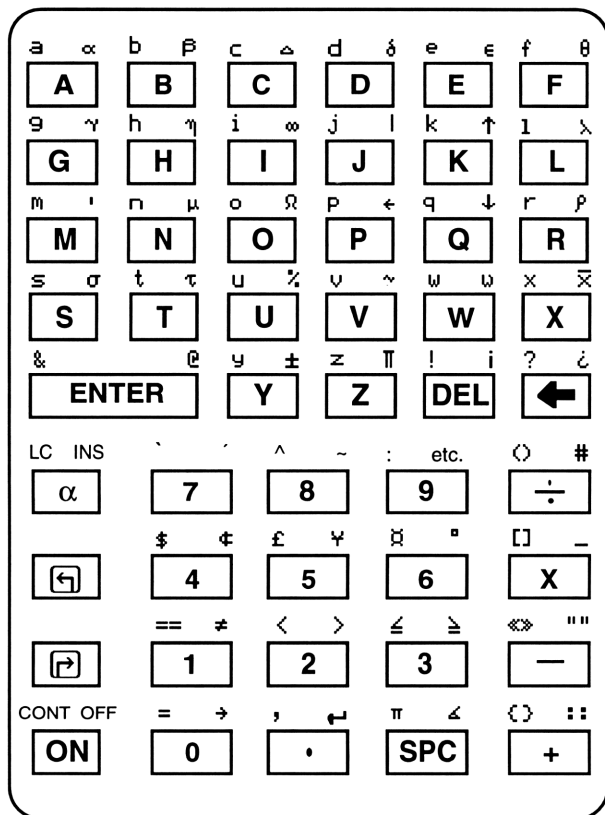
Percent total of x that is represented by y

$x \ y \rightarrow 100y/x$
 $x \text{ 'symb' } \rightarrow \text{'%T(x,symb)'}$
 $\text{'symb' } x \rightarrow \text{'%T(symb,x)'}$
 $\text{'symb}_1 \text{ 'symb}_2 \rightarrow \text{'%T(symb}_1,\text{symb}_2)'$
 $x_unit \ y_unit \rightarrow 100y/x$
 $x_unit \text{ 'symb' } \rightarrow \text{'%T(x_unit,symb)'}$
 $\text{'symb' } x_unit \rightarrow \text{'%T(symb,x_unit)'}$

Related Flag: -3

Note: Units must be dimensionally consistent

Alpha Keyboard



Program Index

ADDCHR	Adds a character object to the end of a string	152
ALGSTK	Uses →LCD to put an algebraic in the stack display	68
AMPLT	Amortization plot	110
ANIM	Demonstrates graphics animation	32
BAKDIFF	Compares iterates of Baker's function	126
BAKER	Plots Baker's function	124
BROWNLN	Draws Brownian line	117
CHKARGS	Checks stack arguments	144
CHOS%	CHOOSE box selects percentage calculations	108
CHOS1	CHOOSE example	64
CHOS2	CHOOSE example	65
CKKB	Checks the key buffer	193
CMP48	Streamlines a program that uses SYSEVALs	196
COERCE	Converts a real number into an internal binary integer	146
COL82	Prints graphics pattern on the HP 82240 printer	218
CRCE2	Converts two real numbers into internal binary integers	147
DAT2STR	Converts date to string	94
DAT2YMD	Converts date to year, month, and day numbers	92
DOW	Finds the day of the week given a date	93
DSORT	Enhanced directory sort	89
DSPER	Displays extended real numbers	165
ENTROPY	Plots proportion of gases in free expansion example	120
ERCALC	Extended real number calculating environment	166
ERTOR	Converts an extended real number to a real number	158
ETOR2	Converts two extended real numbers to real numbers	159
FACTRL	Illustrates recursion	69
FREXP	Graphic illustration of free expansion of gas	122
FUN	Illustrates use of example program XYGD	185
GCD	Greatest common divisor	80
GETMSG	Gets a message from a built-in message table	153
GETVER	Gets version of operating system and model designation	140
GREX	Illustrates graphics operations	28
INF1	INFORM example	62
INF2	INFORM example	63
INPLOT	Plots inequalities	114
JDAYN	Converts date to Julian day number	90
JDCAL	Converts Julian day number to date	91
JULIA	Plots points on the Julia set	135

LBLPICT	Labels <i>PICT</i> with a boxed label	188
LCLV	Illustrates use of \rightarrow as it stores local variables	43
LCM	Least common multiple	81
LINES	DISP example	67
LOGB	Log to an arbitrary base using extended real numbers	162
LREV	Reverses a list using meta-object programs	78
LSTSTR	Searches a string backwards from given position	155
MAH	Adds an object to the head of a meta-object	75
MAH2	Adds an object to the head of a meta-object in pos 2	75
MAM2	Concatenates two meta-objects	76
MANDEL	Draws the Mandelbrot set	138
MAT	Adds an object to the tail of a meta-object	75
MAT2	Adds an object to the tail of a meta-object in pos 2	75
MAVG	DOSUBS example – computes moving average	72
MDH	Extracts object from the head of a meta-object	76
MDH2	Extracts object from the head of a meta-ob in pos 2	76
MDT	Extracts object from the tail of a meta-object	76
MDT2	Extracts object from the tail of a meta-ob in pos 2	76
MENUEX	Custom menu example	229
MKBOX	Creates a blank grob with a border	186
MKERTO A	Creates the program ERTOA	164
ML2M	Converts two lists into meta-objects	77
MM2L	Converts two meta-objects into lists	77
MNU%	Custom menu of percentage calculations	107
MSGBX	Displays message box with graphic icon	199
MSWAP	Exchanges two meta-objects	77
MZ2	Places empty meta-object in position 2	77
NXTSTR	Searches string at specified starting position	154
ORBIT	Plots chaotic orbits	128
ORDXY	Orders points for internal line drawing objects	189
PARSE	Parses a string	156
PASCAL	Displays Pascal's triangle	112
PFIT	Polynomial curve fit	83
PRMCNT	Illustrates use of PROMPT with a custom menu	59
QRT1	Local variable example 1	44
QRT2	Local variable example 2	45
QRT3	Subprogram example 1	46
QRT4	Subprogram example 2	47
QRT5	Subprogram example 3	47

RANWALK	Brownian motion in two dimensions	118
RTOE2	Converts two real numbers to extended real numbers	158
RTOER	Converts a real number to an extended real number	158
SFILT	Filters a list using meta-object programs	78
SHWBX	Places box with border in the stack display	186
SHWP	Polynomial fit with graphic view	84
SLOPE	INFORM example application – calculates line slope	86
SPACES	Creates a string of spaces	153
SQPQ	Square root's partial quotients	82
STKV	Displays up to ten levels of the stack	33
SUBR	Subroutine example	47
TANK	Tank battle on the status line	95
TOCHR	Creates a character object given a real number	150
TOTXT	Completely clears the stack display	181
TRAIN	Steam train crossing the status line	98
TREE	Draws a fractal tree	132
TRPCN	Shows how to trap CANCEL	55
TRYMBX	Illustrates use of example program MSGBX	200
TVMCALC	Customized solver example implements TVM equation	102
TXTBX	Places text in a graphics box	187
TXTEX	Illustrates object to graphics conversions	30
UCRC2	Converts two internal binary integers into real numbers	147
UDFUI	Creates INFORM interface for user-defined functions	88
UNCOERCE	Converts an internal binary integer into a real number	146
VSCAN	Scan input vector using meta-object programs	79
WCM	Illustrates WAIT with a custom menu	60
XYGD	Places a graphics object in the stack display	185
YMD2DAT	Converts year, month, and day numbers to date	93

HP 48 Handbook Disk

Armstrong Publishing Company offers a disk containing all the named example programs in *The HP 48 Handbook*. If you have access to an IBM-compatible or Macintosh personal computer, you can transfer these programs to your HP 48G/GX using the wired serial port, saving time and avoiding potential keystroke errors. See the next page for ordering information.

The HP 48 Pocket Book

The HP 48 Pocket Book goes where you go – providing a concise collection of handy tables covering many aspects of HP 48 operations and a 32 page abbreviated command reference. At 3.5" x 6.375" and 56 pages, *The HP 48 Pocket Book* is small enough to live in your purse or backpack. See the next page for ordering information.

The HP 48 File Manager

The HP 48 File Manager has been designed to simplify data transfer to or from an IBM or IBM-compatible computer.

{ HOME }		
K	V	PRTPAR
↵ E	↵ P	CST
↵ ASTRO	↵ TK	IDPAR
SEND CROIR INFO MARK DISK QUIT		

The arrow keys, **[ENTER]**, **[↶] (UP)**, and **[↷] (HOME)** traverse the directory structure of either the HP 48 or the server disk. Menu keys may be used to show the directory of the Kermit server's disk, transfer files, purge or rename files, create directories, show available free space, or archive the memory of the HP 48. *The HP 48 File Manager* is distributed on a disk for IBM-compatible computers, and includes a free copy of the latest version of Kermit. You'll need a serial cable to connect the HP 48 and the computer (see *Data Transfer*). See the next page for ordering information.

A Modern Difference Engine

Nineteenth century inventor Charles Babbage (1791–1871) is famous for his designs of large mechanical calculating machines – the *Difference Engines* and the *Analytical Engine*. The Difference Engines were designed to subtabulate and print tables of polynomial approximations to log, trig, and other functions.

In 1991 the Science Museum in London completed construction of Difference Engine No. 2 – a modern realization of one of Charles Babbage's never-fulfilled designs. The machine weighs 3 tons and has 4000 parts. Inspired by this machine, software simulators for the HP 48 and PCs have been written to provide you with a chance to experiment with problems that could be put to the Engine. *A Modern Difference Engine* explains the history and designs of these machines, and provides instructions and examples for operation of the simulators.

Ordering Information

Qty	Title	Price	Total
	HP 48 Handbook Disk (IBM)	\$8	
	HP 48 Handbook Disk (Macintosh)	\$8	
	The HP 48 Pocket Book	\$7	
	HP 48 File Manager (IBM only)	\$16	
	A Modern Difference Engine (IBM only)	\$20	
Shipping: Add \$2.00 for each item:			
Total			

Armstrong Publishing Company
1050 Springhill Drive
Albany OR 97321 USA

Checks should be payable to *Armstrong Publishing Company*. Orders originating outside the United States *must* be paid in U.S. dollars with a check drawn on a U.S. bank or paid with an international postal money order. Do not send checks drawn on non-U.S. banks.

The HP 48 Handbook

Do you want to know more about your HP 48? *The HP 48 Handbook* is the ultimate reference for people who want to get the most from the HP 48 calculator. *The HP 48 Handbook* discusses fundamental concepts, programming techniques, user interface tools, graphics, data transfer, and memory management. Additional reference tables cover the equation library, constants, keys, menus, units, and much more.

Over 100 example programs are included to demonstrate various programming and graphics techniques. These programs are designed to show the HP 48 at its best – illustrating everything from financial calculations to fractals to polynomial curve-fitting.

The *Command Reference* lists complete stack diagrams, keystroke access, characteristics, and related flags for every command.

Do you want to use the internal operating system that HP doesn't tell you about? Place graphics in the stack display? Perhaps work with extended precision real numbers? Maintain total control over the keyboard? The *System Programming* chapter introduces the magical world of internal unnamed objects accessible with SYSEVAL.

Join the thousands of satisfied programmers who depend on *The HP 48 Handbook* to get the most from their calculator!

Contents

Introduction	1	Printer Control	217
Objects, Names, and Constants	3	Messages	220
Memory	16	Menus	228
Graphics	24	User-Defined Keys	233
Programming	40	Flags	234
Example Programs	80	Built-In Units	237
System Programming	139	Equation Library	242
System Operations	201	Command Index	249
Statistics Data	204	Command Reference	265
Character Codes	206	Alpha Keyboard	338
Data Transfer	208	Program Index	339