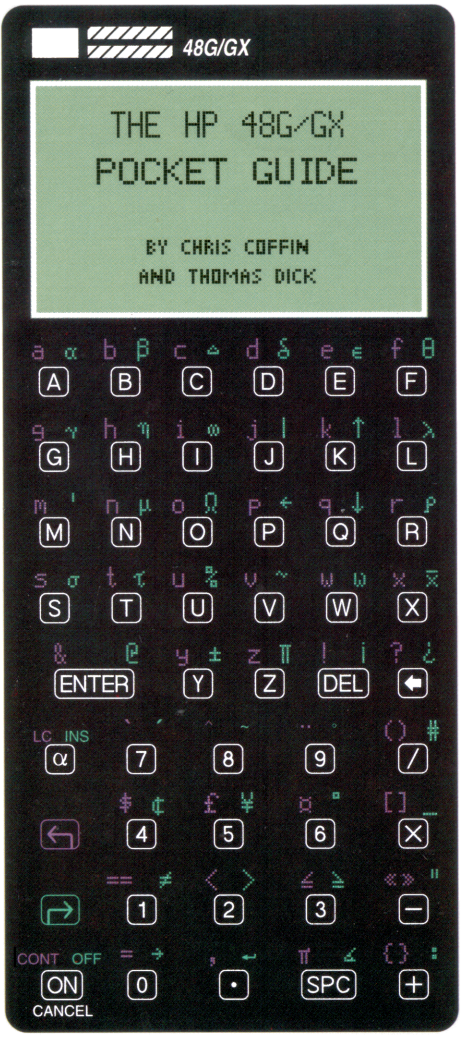


Handy Help—Fits in the Case!

- *Easy look-up: Flags, Objects, Modes, etc.*
- *Examples of applications (PLOT, SOLVE, etc.)*
- *Quick tips and troubleshooting*
- *Menu maps and complete Command Index*

CONTENTS

Alpha Keyboard ..... Front Cover
Object Types and Syntax 1
The Command Line and the Stack 2-4
Arithmetic and Postfix Notation 5
VAR, Variables and Names 6
Reserved Names and Constants 7
MEMORY, Directories and Paths 8-9
Menus 10
MODES 11
CHARS 11
PLOT 12-14
SOLVE 15
UNITS 16-17
MTH 18-19
EQUATION 20
SYMBOLIC 21-22
∂ and Differentiation 23-24
∫ and Integration 24-25
MATRIX, STAT and Σ 26-27
PRG 28-30
TIME 31
I/O and Data Communications 32-33
LIBRARY, Ports and Backups 34-35
EQ LIB and the MES 36-37
CST and Custom Menus 38
Custom Key Assignments 38-39
System Information 39
Frequently Asked Questions 40-41
Command Reference 41-77
System Flags 78-80
Other Products ....Inside Back Cover
Operator Precedence ..... Back Cover
Index ..... Back Cover

Printed in the U.S.A
ISBN 0-931011-45-0
First Printing – April 1996

OBJECT TYPES AND SYNTAX

#	Type	Example
0	Real Number	-6.02E23
1	Complex Number	(.5,-1.57)
2	String	"Hi there!"
3	Real Array	[[1 2] [3 4]]
4	Complex Array	[[(1,0) (.5,-.5)] [(-.5,.5) (0,1)]]
5	List	{ π 3.14 "PI" }
6	Global Name	X
7	Local Name	j
8	Program	« T 11 / »
9	Algebraic Object	'4*π*r^2'
10	Binary Integer	# EFAC11h
11	Graphics Object	Graphic 131 x 64
12	Tagged Object	Answer: 42
13	Unit Object	6_hr/dy
14	XLIB Name	XLIB 543 8
15	Directory	DIR ... END
16	Library	Library 440: ...
17	Backup Object	Backup MYDIR
18	Built-in Function	SIN
19	Built-in Command	CLEAR
20	Int. Binary Integer	<123d>
21	Extended Real #	Long Real
22	Extended Cmpl. #	Long Complex
23	Linked Array	Linked Array
24	Character Object	Character
25	Code Object	Code
26	Library Data	Library Data
27	External Object	External
28	External Object	External
29	External Object	External
30	External Object	External

To test an object's type, put the object onto the stack and execute TYPE.

To test a *name*d object's type, key in its '*name*' and execute VTYPE.

For a list of all objects of a given type in the current directory, key in the desired type# and execute TVARS.

For a list of all objects in the current directory, execute VARS.

The Command Line and the Stack

Entering Objects onto the Stack

To perform arithmetic and other operations on an object (of most any type), you must put the object onto the stack. To accomplish this, you first build it on the ***command line***, then press ENTER, which puts it onto the stack at the bottom (level 1) pushing all other stack objects up a level, and clearing the command line for the next building session.

If you err while typing on the command line, use ◀ or ▶ to move to the offending character(s), then DEL or ⬅ to erase them. Or abort the whole thing via CANCEL (ON). Or, use the ↶ EDIT menu and its various other cursor movement tools and editing options:

↶ EDIT **(28)***

←SKIP SKIP→ ←DEL DEL→ INS ■ ↑STK

SKIP→ skips the cursor to the far right; **DEL→** *deletes* everything to the right; likewise to the left for **←SKIP** and **←DEL**. If you toggle the **INS** key *off* (so that the white square disappears from the menu item), the cursor becomes square and will type over characters rather than inserting between them.

Delimiters

Bear in mind that you can actually build more than one complete object on the command line prior to pressing ENTER. To separate successive objects, put a *delimiter* character between them. For example, by typing 1 SPC 2 SPC 3 SPC 4 SPC 5 ENTER, you would be entering five objects onto the stack at once; each would go on in the order you typed it. The most common "all-purpose" delimiter characters are the comma and the space (as used here), but the syntaxes of many specific object types (see page 1) include other distinctive beginning/ending characters that can serve as delimiters, too.

The Shift Keys

When typing on the command line, you'll soon discover that there are far more commands and characters available for entering objects than there are act-

*Menu numbers, such as this **28**, are discussed on page 10.

ual keys. The machine therefore uses 3 ***shift keys*** to alter the meanings of many keys: [←], [→] and [α].

The purple [←] key changes most keys' functions to those labelled in purple above the keys; likewise for the [→] key and the green labels.

[α] transforms the keys to the ***alpha keyboard***, which offers access to alphabetic characters and other symbols (some appearing at the lower right of the keys).*

Any two of the three shifts may be in effect at any time, except [←]-[→], thus extending the keyboard's functional range nearly six-fold. Note that when any shift is in effect, its symbol appears in the *annunciator area* at the top of the display, which gives many such status signals (e.g. ⧖ means "calculation in progress").

Note also that a shifted mode usually lasts *only one keystroke* (e.g. you may press [←] or [→], then just one other key before that shift mode ends)—unless you hold the shift key down. But pressing [α][α] will *lock on* alpha mode; it then takes a third [α] to unlock it.

Editing an Object Already on the Stack

To alter an object that is already entered on the stack (level 1), just press [←][EDIT]—without starting a command line first. This sends the level 1 object to the command line for modification—which you then do exactly as if you had just now keyed it all in. After making the changes you want, press [ENTER] to place the edited version back at stack level 1. If you botch the editing process and would prefer to restore the original level 1 object, [ON] ([CANCEL]) aborts the edit.

Altering the Stack—the Common Commands

[←][SWAP] exchanges objects on stack levels 1 and 2.**

[←][DROP] erases the object on level 1 and drops all remaining objects down a level.**

[←][CLEAR] clears the stack of all objects.**

When the command line is clear, [ENTER] does a DUP command—duplicating the level 1 stack object onto level 2 (and pushing everything else up a level).

*See page 11 for more about the various characters and how to type them. Note also the reference keyboard on the front cover of this book.

**To save keystrokes with this command, the [←] key is *unnecessary* if the command line is clear (i.e. if [▶], [◀] and [DEL] are inapplicable).

Most other stack commands are various exchanges, shifts or duplications, which you can do two ways.

The Interactive Stack

[→][MENU] (or just [▲]) starts the ***interactive stack***, where you can move a pointer (using [▲] and [▼]) to any given level. For example, press [←][CLEAR] and do this: [3][ENTER] [1][ENTER] [7][ENTER] [8][ENTER] [5][ENTER]. The 3 you entered is now out of view, up at level 5. To move up the stack to see it, press [▲] to activate the interactive stack pointer, then just [▲][▲][▲][▲].

Notice the menu items that appear in the interactive stack. Move your pointer down to level 1 ([▼][▼][▼][▼]), then up to level 3 ([▲][▲]). Now try these menu items: **VIEW** sends the contents of the current level to the editor, where you can edit it without disturbing other objects on the stack. Use [ENTER] to accept the change (or [CANCEL] not to). **PICK** copies the current level's object down at level 1; all other entries will shift up a level (now you should have *six* entries). **ROLL** sends the current level's object to level 1; the entries below it shift up one level. **ROLLD** sends the current level's object down a level; the previous contents of level 1 take its place. **ECHO** "echoes" the object at a stack level to the command line. (The **+STK** item in the [←][EDIT] menu also starts the interactive stack to let you use **ECHO**.) [CANCEL] returns to the normal stack.

[←][STACK] (73)

OVER	ROT	ROLL	ROLLD	PICK	DEPTH
DUP	DUP2	DUPN	DROP2	DRPN	

The [←][STACK] menu offers stack manipulations similar to the interactive stack, but you indicate the desired level with a *command argument*, not a pointer.

Error Recovery Features*

The [→][UNDO] operation (also commonly known as "Last Stack") returns the stack to its status just before the last operation. Other related commands: [→][ARG] keeps your last result on the stack but returns the last arguments used by the HP 48G/GX. [→][CMD] offers the last four command line entries for retrieval.

*Because these features consume time and memory to maintain, they can be deactivated—see the [←][MODES] menu under **MISC**.

Arithmetic and Postfix Notation

"Crunching" Numbers on the Stack

Every good calculator should do arithmetic quickly and easily. HP calculators pioneered a very efficient style of operation called ***postfix notation*** (a generalized version of the method known as "RPN"), in which you put the arguments on a stack first, then perform the operation last. This precludes the need to use parentheses—any intermediate results simply "float" in the stack until needed. Here are some examples:

Add $26 + 82$: [2][6][ENTER][8][2][+]
Subtract $86 - 32$: [8][6][ENTER][3][2][−]
For a negative number, use [+/−]: [2][+/−][ENTER].
Multiply 62×45: [6][2][ENTER][4][5][×]
Divide $85 \div 20$: [8][5][ENTER][2][0][÷]
Raise 42 to the 5th power: [4][2][ENTER][5][y^x]
Take $\sqrt{20}$: [2][0][$\sqrt{x}$]
Square 25: [2][5][←][x^2]
Find the reciprocal of 85: [8][5][$1/x$]
Calculate $(5+2)^3$: [5][ENTER][2][+][3][y^x]
Calculate $5 + 2^3$: [5][ENTER][2][ENTER][3][y^x]
Calculate $6\sqrt{10}$: [6][ENTER][1][0][$\sqrt{x}$][×]
Calculate $(21+7)^{4/5}$: [2][1][ENTER][7][+][4][ENTER][5][÷][y^x]
Find $\sin(25)$: [2][5][SIN]
Find $\arctan(1)$: [1][←][ATAN]
Find $\log(2)$ (*common* log—base 10): [2][→][LOG]
Find $\ln(3)$ (*natural* log—base e): [3][→][LN]
Find e^{10}: [1][0][←][e^x]

Postfix Notation—Everywhere in the Machine

Again, note the postfix notation method in the above calculations: ***Arguments first, operation last.*** And note also that the ***arguments are consumed*** by the operation. For example, after adding 26 and 82, you get only the result (108) on the stack; the arguments, 26 and 82 are gone. *This is true in general for operations throughout the calculator* (whether or not you expect any stack results): If an operation uses any arguments, you put them on the stack prior to executing the operation. The arguments are then consumed (regardless of results). For example, to set the display to 2 decimal places, you use the FIX command, with 2 as the argument: [2][ENTER][α][α][F][I][X][α][ENTER]. The argument is consumed (and, in this case, no new result objects appear on the stack).

VAR, VARIABLES AND NAMES

Storing Objects in Names

If an object is on the stack, you can ***name*** it (i.e., ***store*** it in a name): Position the object at stack level 2 and the desired name at level 1, using algebraic apostrophes around the name (e.g. '*name*'). Then just press STO. Name and object (the arguments) disappear from the stack, but the name appears as the newest (left-most) item in the **VAR**iable ***menu***, which you see via VAR. This menu (which is menu #2) shows the names of all objects stored in the current directory.

Examples: To store the real number 93 in the name QUIZ, press 9 3 ENTER ' α α Q U I Z α STO.

To store the vector [3 4] in the name TRI, press ↶ [ ] 3 SPC 4 ENTER ' α α T R I α STO.

To store the algebr. expression 'L*W' in Area, press ' α L × α W ENTER ' α α ↶ α ↶ A R E A α STO.

To store the name QUIZ in the name TEST, press ' α α Q U I Z ENTER ' α α T E S T α STO.

Invoking Names

In any menu, including VAR, a menu item is just a typing aid, a shortcut for typing the name and pressing ENTER. So pressing a VAR menu key treats that name like any other name—it *evaluates* it, putting its contents onto the stack (or executing it, if it's a program). So, after the above examples, pressing QUIZ (or typing α α Q U I Z ENTER) will put 93 onto the stack. By contrast, to put a name onto the stack *without* evaluating it, you must enter it as an algebraic object: ' QUIZ (or ' α α Q U I Z α) ENTER.

Note also that you can use ↶ to store and ↷ to recall VAR name values. So (again, using the above), 9 9 ↶ QUIZ is the same as 9 9 ENTER ' QUIZ STO. And ↷ QUIZ is the same as ' QUIZ ↷ RCL.

Purging (Erasing) Names

To purge a name from the VAR menu in the current directory, put the '*name*' on the stack and use ↶ PURG. To purge a group of names at once, put the names into a list (and the VAR menu is a typing aid here, too): ↶ { } QUIZ AREA..., etc., ENTER), then use ↶ PURG.

Using Names Wisely

Names may not contain delimiter characters nor begin with numerals (0-9), but they may be quite long and include lowercase letters and other non-alphabetic characters. But the VAR menu can *display* only the first 4-5 characters in each name—and only uppercase representations of letters. So let your names be distinct (and descriptive) within the first few characters. For example, TEST and Test are both valid names, but they'll both appear as TEST on the VAR menu. And OBJECT1 and OBJECT2 are both valid names, but they'll both appear as OBJEC.

Reserved Names and Constants

You should avoid using some names for general-purpose storage, because their contents are *interpreted* by the calculator for certain specific purposes:

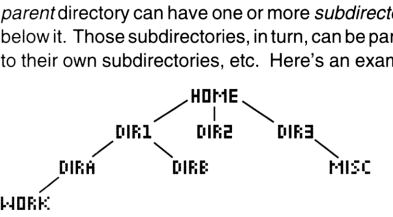
ALRMDAT	Current data for current alarms.
CST	Current contents of custom menu.
EQ	Current equation used by SOLVE & PLOT.
EXPR	Current expression, symbolic operations.
e	(Symbolic constant*) 2.71828182846.
IERR	Uncertainty in current integration.
IOPAR	Current parameters for I/O operations.**
i	(Symbolic constant*) (0, 1).
MAXR	(Symb. const.*) 9.99999999999E499.
MHpar	Data of currently saved Minehunt game.
MINR	(Symbolic constant*) 1.E-499.
Mpar	Current equation set for Mult. Equ. Solver.
Nmines	Current number of mines in Minehunt.
PICT	Current contents of graphic display.
PPAR	Current parameters for plot operations.
PRTPAR	Current parameters for print operations.**
VPAR	Current parameters for viewing 3-D plots.
ZPAR	Saves previous PPAR for zoom operations.
der...	Indicates a user-defined derivative.
n1, n2, ...	Integer coefficients used by ISOL.
s1, s2, ...	Sign coefficients used by ISOL and QUAD.
π	(Symbolic constant*) 3.14159265359.
ΣDAT	Current matrix of data used for statistics.
ΣPAR	Current parameters for statistics calc.

*Contents of this name *cannot* be changed (STO gives an error); display result depends on Flags -2 and -3 (see page 78).

**Interpreted this way in the HOME directory only.

MEMORY, DIRECTORIES AND PATHS

The machine's memory is organized into a hierarchical *tree* of directories, which you construct. Each *parent* directory can have one or more *subdirectories* below it. Those subdirectories, in turn, can be parents to their own subdirectories, etc. Here's an example:



To build this example structure, you'd start at the built-in **HOME** directory (press → HOME), and use CRDIR to create the other directories (as with most commands involving names, you enter either a { *list* } or one '*name*' at a time): ← {} α α D I R 1 SPC D I R 2 SPC D I R 3 ENTER α α C R D I R ENTER. Those three directories now appear in the VAR menu of your **HOME** directory (press VAR). Note how directory menu items resemble tabbed folders. Press **DIR3**. This new directory is empty —nothing stored in it yet. Now create one sub-directory, MISC: ' α α M I S C ENTER α α C R D I R ENTER.

Similarly, create **DIRA** and **DIRB** in **DIR1**: Press → HOME, then **DIR1** to move to that directory. Then: ← {} α α D I R A SPC D I R B SPC ENTER α α C R D I R ENTER. Now press **DIRA** and create one subdirectory, WORK: ' α α W O R K ENTER α α C R D I R ENTER.

The *current directory* (where STO puts named objects) is simply your current location, which you change by moving about in the tree. (← UP moves you up a level at a time.) Note how the annunciator area displays the complete *path* of the current directory (and the PATH command will produce that path list on the stack for handy storage and/or quick navigation later).

A directory structure helps organize your information, but more importantly, it helps to shield named objects from accidental "clobbering" during unrelated operations. When given any name to evaluate (execute), the HP 48G/GX first looks in the current directory for that name. If it doesn't find it there, it looks *upward*, one level at a time, *along the path* (toward **HOME**). So any named object stored in a directory *in the path* of

the current directory is "visible" (usable) from the current directory; all other names are "invisible." Thus, in the example, all names in **DIR1** are also visible from **DIRA**, **DIRB** and **WORK**, but *not* from **DIR2**, **DIR3**, **MISC**, or **HOME**. And, as with the built-in (command) names, all names in **HOME** are visible from all directories.

To erase an empty directory, you can use the keyboard command ◀PURG (just as for other named objects). But to avoid grievous errors, for a *non*-empty directory, you need the *non*-keyboard PGDIR command. Thus, to erase the example structure, first go **HOME** (▶HOME) then put { DIR1 DIR2 DIR3 } on the stack and use PGDIR. (You could use ◀PURG on 'DIR2' alone, but it's easier to do it all at once.)

▶MEMORY

The ▶MEMORY application offers a self-contained input screen that allows you to move, copy, edit, create or purge any named object in memory. As with most such application screens, you use the arrow keys (▲, ▼, ◀ and ▶) to move the highlight bar around, then ENTER or **OK** to accept your selections. You can select more than one object by using the **☑CHK** tool.

◀MEMORY (70)

(71)

(72)

The ◀MEMORY menu has 2 commands for measuring memory storage and data accuracy: MEM measures the free memory (in bytes) left in your machine; BYTES measures the size (in bytes) and accuracy (a binary-integer checksum) of a given object.

This menu also has two main sub-groups: **DIR** is a collection of directory commands. Along with PATH, CRDIR and PGDIR, you get commands that are useful for working with the VAR menu display (ORDER reorders that menu; VARS and TVARS put the menu—or a subset—onto the stack as a list of names). Commands in the other sub-group, **ARITH**, allow you to perform arithmetic directly with on object stored in a name—no need to return the object to the stack first.

MENUS

Menus are typing aids—nothing more. Using a menu key to invoke a name is simply a quick way to enter the name; command line or menu, the result is the same.

Don't confuse menus with directories: Directories are the partitioned areas in memory where object names are actually stored; *menus are just selected groups of name-typing aids collected in the display for your convenience.* The names represented by a menu might actually be stored in various directories (and as noted on pp. 8-9, the result of invoking any name, via menu or otherwise, depends on that name's actual storage location in relation to the current directory path).

The VAR menu is often confused with a directory, because its typing aids are for all the names stored in the *current* directory; if you move to a different directory, you see a different VAR menu. This gives the (false) impression that, to invoke a name, you must be in the directory where that name is stored. **No**. You can invoke any name (typed "longhand" or via menu—no difference), as long as that name is stored somewhere in the current directory *path*. For example, all built-in command names reside, in effect, "at **HOME**," so that you can indeed invoke them from anywhere. (If invoking a name required you first to move to its actual storage location, you'd have to press [→][HOME] before invoking any built-in name—not very handy.)

Menus can have more than one "page" (i.e. more than six typing aids); you reach the other page(s) via [NXT] or [←][PREV]. There are over 100 pre-defined menus (and you can build your own, too—see p. 38). All predefined menus are numbered for use with the MENU command. (Those numbers are shown herein beside each menu diagram.) For example, the 2nd page of the [←][CHARS] menu (#62—see opposite) is denoted 62.02. So, instead of moving there via [←][CHARS][NXT], you can use 62.02 as the argument for MENU.

One final thought: The keyboard itself is also a sort of menu. It is, after all, just another collection of typing/entering shortcuts. But notice this about its design: The [→]-shifted version of a key often gives a handy, fill-in-the-blanks *application screen*; and the [←]-shifted version of the same key offers a pre-defined *menu* of related programmable *commands*.

MODES

The [→][MODES] screen lets you adjust machine modes (some signalled in the annunciator area). You select options via **CHOOS**, **✔CHK** (or [NXT] **RESET**) and especially **FLAG**, which explains each mode.

[←][MODES] **(63)**

(64) **FMT**

STD ■	FIX	SCI	ENG	FM.	ML ■
					MODE

(65) **ANGL**

DEG ■	RAD	GRAD	RECT■	CYLIN	SPHER
					MODE

(66) **FLAG**

SF	CF	FS?	FC?	FS?C	FC?C
STOF	RCLF	RESET			MODE

(67) **KEYS**

ASN	STOK	RCLK	DELK		MODE

(68) **MENU**

MENU	CST	TMEN	RCLM		MODE

(69) **MISC**

BEEP■	CLK	SYM■	STK ■	ARG ■	CMD■
INFO■					MODE

The [←][MODES] menu simply offers an **ON ■/ OFF** format for setting the various calculator modes shown.

CHARS

[→][CHARS] offers screens showing the ASCII **NUM**ber (**0-255**) and keystroke (if any) of all machine characters (and note that this book's front cover offers a partial reference). **-64** and **+64** let you move between the four screens. You browse via the highlight box and arrow keys, and you can **ECHO** any character to the command line if you wish. [CANCEL] exits.

[←][CHARS] **(62)**

SUB	REPL	POS	SIZE	NUM	CHR
OBJ→	→STR	HEAD	TAIL		

The [←][CHARS] menu offers commands that are useful in building, analyzing and dissecting a character string, including such things as finding its length, extracting or substituting substrings within it, and removing its first character. For more details on any particular command, see the Command Index (pp 41-77).

PLOT

→PLOT lets you draw some 15 types of mathematical plots, including simultaneous and 3-D plots.

Example: Plot $\tan x$. Press →PLOT, highlight **TYPE:** and press **CHOOS** to see your choices.

1. Highlight Function and press **OK**.
2. Set radians mode (highlight ∡:; **CHOOS** or +/–).
3. Reset default parameters (center on the origin, use coordinate axes, and put an axis mark every 10 pixels, representing 1 unit): Press NXT **RESET**, select Reset plot and press **OK**.
4. Next, highlight **EQ:** and type ' TAN α X ENTER. Now, you can leave the default viewing scale or **☑CHK** **_AUTOSCALE** (which samples x-values to estimate a y-range), or enter specific vertical and horizontal ranges. For now, leave the defaults.
5. Press NXT **ERASE** (to clear prior plot), and **DRAW**.

After a first plot, you'll want to adjust it at the **OPTS** screen (press **CANCL** to see the main **PLOT** screen). At **PLOT OPTIONS**, you set the *calculation* (not the view) range for the independent variable (the *default* is the view range). You can also set *connected* mode: If **_CONNECTED** is unchecked, only 1 pixel per column is lit on the plot; if **_CONNECTED** is checked, additional pixels are lit to give the graph a continuous look.* (Try repeating the above with **_CONNECTED** *un*checked.)

Interactive Graphing Features

Once you've drawn a plot, the arrow keys move a set of crosshairs.** +/– toggles the crosshairs to "invert" (white-on-black) if needed. Press **CH.YP** or + and crosshair *coordinates* appear below (press any menu key to return the menu labels). Use **TRACE** to put the crosshairs on the graph. Press **CH.YP** again for coordinates of a point on the graph.

Look more at the interactive menu: The **ZOOM** folder offers tools for rescaling your view. **BOX** lets you draw a box to show a desired viewing window; **ZFACT**

*In the example, it only *appears* that "vertical asymptotes" have been drawn; the plotter has just connected the pixels *across* the asymptotes. Note, however, that if such a discontinuity falls *exactly* on the coordinate of a column of pixels, the graph won't connect over the discontinuity.

**ENTER puts the crosshairs coordinates on the stack, but *note:* those are *pixel coordinates*—not necessarily a point on your function graph.

lets you set horizontal and vertical zoom factors (the default is 4), then **ZIN** *zooms in* by these factors; **ZOUT** *zooms out* (or you can scale each axis separately via **HZIN**, **HZOUT**, **VZIN** and **HZOUT**). **ZSQR** *squares* the view (so a pixel is the same scale horizontally as vertically); **ZDFLT** resets to default viewing ranges ($[-6.5,6.5] \times [-3.1,3.2]$); **CNTR** *centers* the screen on the crosshairs; **ZAUTO** *autoscales* the vertical range. **ZDECI** sets x-axis "tick" marks 1 unit apart (so a pixel = .1 units); **ZINTG** sets them 10 units apart; and **ZTRIG** $\pi/2$ units apart. **ZLAST** is the "undo zoom" key; **PICT** restores the interactive graphics menu.

The **FCN** folder has interactive tools for analyzing functions: **ROOT** puts the crosshairs on the nearest root, displays it, and tags it (on the stack); **SLOPE** calculates the derivative of the function at the crosshairs, displays it, and tags it. Press **AREA** once to mark the lower limit of integration at the crosshairs (use ⊠ to move this "mark"); press **AREA** again to compute the definite integral from that lower mark to the current crosshairs position, display it, and tag it (integrating right-to-left gives the negative of left-to-right). **EXTR** snaps the crosshairs to the nearest extremum, displays it, and tags it. **F(X)** finds the function's value at the crosshair location, displays it, and tags it. Use **F'** to find the function's derivative and graph it with the original function; press **F'** again to plot the *2nd* and 1st derivatives, and the function; etc.

Plotting Multiple Functions Together

To plot more than one function together on the same graph, enter them into the **EQ:** field as a *list* (example: `{ 'TAN(X)' 'SIN(X)' }`). If **SIMULT** is checked in the **PLOT OPTIONS** screen, all functions are drawn in one sweep; if not, they're drawn in successive sweeps.

Plotting "Split-Defined" Functions

A "split-defined" function's definition depends on the input. *Example:* `'IFTE(X≤1,X^2,1-X)'` ("*If* $x \leq 1$, *then* use x^2, *else* use $1 - x$") Try it as a **Function** plot—default settings (but disable connected mode).

Other Plot Types

Use a **Truth** plot type to shade regions that satisfy ***inequalities***. *Example:* `'SIN(X*Y)≥.5'` (Use default parameters—and be patient.)

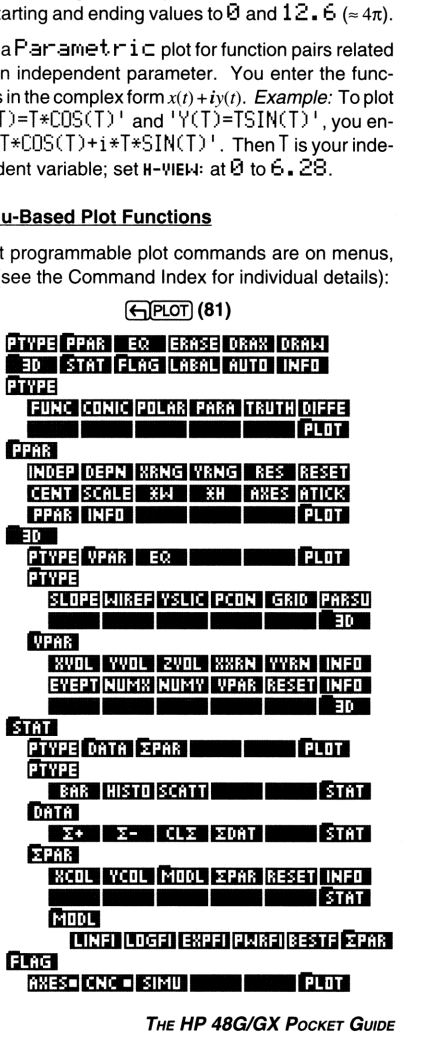
Use a Conic plot type for circles, hyperboli, paraboli, or ellipses. *Exmpl.:* '4*X^2-3*X*Y+Y^2-4=0'. View from −6.5 to 6.5 (H:) and −3.2 to 3.1 (V:).

Use a Polar plot type for functions dependent on a central angle. *Example:* 'R=3*SIN(2.5*θ)' To plot this, change the independent variable to θ and set its starting and ending values to 0 and 12.6 (≈ 4π).

Use a Parametric plot for function pairs related by an independent parameter. You enter the functions in the complex form $x(t)+iy(t)$. *Example:* To plot 'X(T)=T*COS(T)' and 'Y(T)=TSIN(T)', you enter 'T*COS(T)+i*T*SIN(T)'. Then T is your independent variable; set H-VIEW: at 0 to 6.28.

Menu-Based Plot Functions

Most programmable plot commands are on menus, too (see the Command Index for individual details):



SOLVE

↱SOLVE will numerically* solve a general algebraic equation, a differential equation, or a linear system (and two other specific kinds of algebraic equations—polynomials and compound interest). In each case, you key in either the equation or its coefficients—as it equates to zero—then solve for the variable(s).

Example: Find all roots of $4x^3-11x-7$. Press ↱SOLVE and select `Solve poly...`. Now key in the coefficients (including zeroes for any missing terms) as a vector: `[ 4 0 -11 -7 ]`. Then, with the highlight on the ROOTS: field, press SOLVE.... The roots will appear as the terms in a vector—either real or complex—both in the ROOTS: field and tagged on the stack. (Note that you can get a *symbolic* version of either the polynomial or its solution(s) on the stack if you press SYMB while highlighting the desired field.)

The linear system option ($A \bullet X = B$) is similar, except that you key in a *matrix* (A) of coefficients, plus a *vector*(B) of constants, then solve for *vector* X.

↰SOLVE (74)

↰SOLVE offers a menu-based solver (called SOLVR). The mechanics are simple: Store your equation in EQ, then start SOLVR. To indicate a known value, just key it in and press the desired menu key; to recall a value, press ↱ and then the desired menu key; to calculate a value, press ↰ before the desired menu key.

(75) ROOT
SOLVR ROOT EQ SOLVE

(30) SOLVR
(menu items will vary with contents of EQ*)*

(76) DIFFE
RKF RRK RKFS RRKS RKFE RSBER
SOLVE

(77) POLY
PROOT PCOEF PEVAL SOLVE

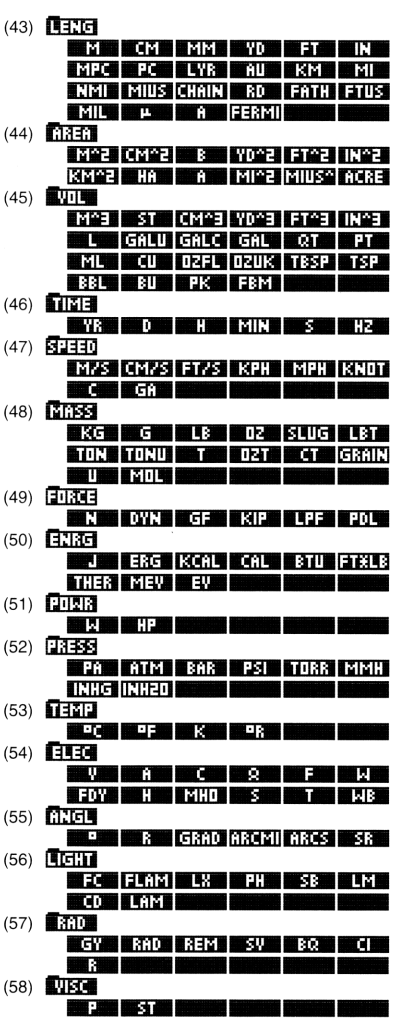
(78) SYS
/ LSQ RSD SOLVE

(79) TVM
SOLVR TVMR AMOR BEG SOLVE

(80) SOLVR
N I%YR PV PMT FV AMRT
PYR BEG

*To solve an equation symbolically for any particular variable, use `Isolate var...` or `Solve quad...` in ↱SYMBOLIC.

→ UNITS (42)


(43) LENG
M CM MM YD FT IN
MPC PC LYR AU KM MI
NMI MIUS CHAIN RD FATH FTUS
MIL μ Å FERMI
(44) AREA
M^2 CM^2 B YD^2 FT^2 IN^2
KM^2 HA Å MI^2 MIUS^ ACRE
(45) VOL
M^3 ST CM^3 YD^3 FT^3 IN^3
L GALU GALC GAL QT PT
ML CU OZFL OZUK TBSP TSP
BBL BU PK FBM
(46) TIME
YR D H MIN S HZ
(47) SPEED
M/S CM/S FT/S KPH MPH KNOT
C GA
(48) MASS
KG G LB OZ SLUG LBT
TON TONU T OZT CT GRAIN
U MOL
(49) FORCE
N DYN GF KIP LPF PDL
(50) ENRG
J ERG KCAL CAL BTU FT*LB
THER MEV EV
(51) POWR
W HP
(52) PRESS
PA ATM BAR PSI TORR MMH
INHG INH2O
(53) TEMP
°C °F K °R
(54) ELEC
V A C Ω F W
FDY H MHO S T WB
(55) ANGL
° R GRAD ARCMI ARCS SR
(56) LIGHT
FC FLAM LX PH SB LM
CD LAM
(57) RAD
GY RAD REM SV BQ CI
R
(58) VISC
P ST
(59) ← UNITS
CONV UBASE UVAL UFACT →UNIT

To attach, convert or combine physical units of measurement, use the →UNITS application:

To ***attach*** a unit object to a real number (or another unit) object, just press the desired unit's menu key. This *multiplies* the given object by the desired unit. *Example:* To form 2.54_cm ("2.54 centimeters"), press 2 · 5 4 →UNITS **LENG** **CM**.

To ***attach the inverse*** of a unit object to a real number (or another unit) object, press → and the desired unit's menu key. This *divides* the given object by the desired unit. *Example:* To form 1.8_°R/K ("1.8 degrees Rankine per Kelvin"), first press 1 · 8 →UNITS **TEMP** **°R**, then just press → **K**.

To ***convert*** a unit object to another (equivalent) unit, press ← and the desired unit's menu key. *Example:* To convert 26.21875_mi ("26.21875 miles") to kilometers, first press 2 6 · 2 1 8 7 5 →UNITS **LENG** **MI**, then ← **KM**. *Result:* 42.194988_km

Addition and subtraction of units require like dimensions. The result units are those of the second argument. *Example:* To add 2 feet and 3 inches, press 2 →UNITS **LENG** **FT** 3 **IN** + (*result:* 27_in). But now reverse the arguments: 3 →UNITS **LENG** **IN** 2 **FT** + (*result:* 2.25_ft).

You can ***multiply or divide*** any mix of units just as in real-number arithmetic. *Example:* To multiply 25 g by 8 ft/s², you would press 2 5 →UNITS **MASS** **G**, 8 →UNITS **LENG** **FT**, →UNITS **TIME** → **S** → **S**, ×. *Result:* 200_g*ft/s^2

To ***convert a mixed-units result to SI base units***, you can use UBASE (available on the ←UNITS menu). *Example:* To convert 200_g*ft/s^2 (continuing from the above example) to SI base units, just press ←UNITS **UBASE**. Result: .06096_kg*m/s^2

The CONVERT command offers another way to convert a unit to any equivalent unit: Given the original unit, just key in the desired unit and use CONVERT.* *Example:* To convert .06096_kg*m/s^2 (continuing from the above example) to the equivalent pounds of force, key in 1_lbf and press ←UNITS **CONV**. *Result:* 1.37043531714E-2_lbf.*

*Thus you really have *three* ways to convert between equivalent units: The →-shifted →UNITS menu items; the CONVERT command; and (a quick trick) adding *zero* of the desired units (e.g. above: 0_lbf +).

MTH (3)

(4) VECTR

ABS	DOT	CROSS	V→	→V2	→V3
RECT▪	CYLIN	SPHER			

(5) MATR

MAKE	NORM	FACTR	COL	ROW	LSQ
RSD	EGV	EGVL	→DIAG	DIAG→	

(6) MAKE

CON	IDN	TRN	RDM	RANM	SIZE
GET	GETI	PUT	PUTI	SUB	REPL
					MATR

(7) NORM

ABS	SNRM	RNRM	CNRM	SRAD	COND
RANK	DET	TRACE			MATR

(8) FACTR

RREF	LU	LQ	QR	SCHUR	SVD
SVL					MATR

(9) COL

→COL	COL→	COL+	COL-	CSWP	MATR

(10) ROW

→ROW	ROW→	ROW+	ROW-	RCI	RCIJ
RSWP					MATR

(11) LIST

ΔLIST	ΣLIST	ΠLIST	SORT	REVLI	ADD

(12) HYP

SINH	ASINH	COSH	ACOSH	TANH	ATAN
EXPM	LNP1				

(14) REAL

%	%CH	%T	MIN	MAX	MOD
ABS	SIGN	MANT	XPON	IP	FP
RND	TRNC	FLOOR	CEIL	D→R	R→D

(15) BASE

HEX	DEC▪	OCT	BIN	R→B	B→R
LOGIC	BIT	BYTE		STWS	RCWS

(16) LOGIC

AND	OR	XOR	NOT		BASE

(17) BIT

RL	SL	ASR	SR	RR	BASE

(18) BYTE

RLB	SLB		SRB	RRB	BASE

(13) PROB

COMB	PERM	!	RAND	RDZ	
UTPC	UTPF	UTPN	UTPT	NDIST	

(19) FFT

FFT	IFFT				

(20) CMPL

RE	IM	C→R	R→C	ABS	ARG
SIGN	NEG	CONJ			

(21) CONS

E	2.718	i	(0,1)	π	3.141
MINR	1.E-4	MAXR	9.999		

Most of the HP 48G/GX's math commands are grouped together under the MTH key. Recall that with most types of objects, you do arithmetic or other operations just as you would with real-number arithmetic: First you put the argument(s) on the stack, then you select the operation; the result replaces the argument(s). Here is a brief overview of the various MTH command groups (but for more details on a particular command, see the Command Index on pages 41-77):

The ***vector*** commands (under VECTR) let you build, calculate (and even change the coordinate system) with 2D, 3D, or n-D vectors (syntax: [ ]).

The ***matrix*** commands (under MATR) let you build and calculate with nxm matrices (syntax: [[ ]]), including various factors, diagonals, least-square calculations and Eigenvalues.

The ***list*** commands (under LIST) offer element-by-element procedures for summing, multiplying, sorting and reversing the elements in a given list (list syntax: { }). (See also list commands under PRG LIST.)

The ***hyperbolic math*** commands (under HYP) offer the 3 basic hyperbolic trig functions and their inverses, plus high-precision versions of LN and EXP.

The ***real number*** commands (under REAL) include percentage and modulo calculations and commands that extract portions of real values (rounding, absolute value, integer and fractional portions, etc.).

The ***binary integer*** commands (under BASE) allow you to build, alter, and do logical calculations with binary integers (syntax: #...)—and choose the format and extent of their display.

The ***probability*** commands (under PROB) calculate permutations, combinations, factorials, pseudo-random numbers, and various probability distributions.

The ***fast Fourier transform*** (FFT) commands do just that—compute fast Fourier transforms.

The ***complex number*** commands (under CMPL) let you build, take apart, conjugate and negate complex numbers (syntax: (,)).

The ***constants*** menu (under CONS) offers typing aids for entering the various values and names of the machine's built-in constants (e, i, π, MAXR and MINR).

EQUATION

Pressing [←][EQUATION] starts the EquationWriter, an alternate form of command line useful when entering algebraic expressions or equations. Instead of the normal command-line approach to expressions (i.e. using the '*expr*' format), which relies on hard-to-read parentheses, the EquationWriter lets you write expressions in a more familiar textbook ("chalkboard") form. So, instead of '1/√(3*A^2+4*B^2)-5*HYP', for example, you'll instead see $\frac{1}{\sqrt{3a^2+4b^2}} - 5Hyp$.

Note, however, that this format is *for your eyes only*—and only while you're working with this command line. When you press [ENTER], the finished algebraic expression will take its place on the stack in the usual '*expr*' syntax, parentheses and all.

The most important key when using the EquationWriter is [▶]. When building an expression, you must use [▶] to move to the next "component" of the expression. That is, you use [▶] to exit a denominator or get "outside" a set of parentheses, a radical sign, an exponent, etc. *Example:* Use the EquationWriter to enter

$$\frac{1}{\sqrt{3a^2+4b^2}} - 5Hyp$$

Start the EquationWriter: [←][EQUATION]. The stack disappears (but not the menu key labels). Now watch your screen as you do these keystrokes: [1][÷][√x][3][α] [A][y^x][2][▶][+][4][α][B][y^x][2][▶][▶][▶][−][5][×][α][α][H][Y][P].

Notice how [▶][▶][▶] exits first the final exponent, then the radical, then the denominator. Note also that you can use *implied multiplication* (i.e. omit the [×] keystroke) *with single-letter variable names only.* When you're ready, [ENTER] sends the expression to the stack (just like the other, "normal" command line). There the algebraic object appears in its usual line format: '1/√(3*A^2+4*B^2)-5*HYP'

To edit an algebraic expression already on the stack, you could press [←][EDIT], as usual, to send it to the normal command line. But to send it instead to the EquationWriter, press [←][VIEW] (or just [▼]) instead.*

*[←][VIEW] is like [←][EDIT], only smarter: Depending on the type of object, [←][VIEW] sends it either to the EquationWriter (for algebraics), the Matrix Writer (for arrays), or to the normal command line (for most other types).

SYMBOLIC

Enclosing an expression in algebraic syntax, ' ', suppresses immediate evaluation of the result; you must press EVAL explicitly. This lets you build and use symbolic expressions with variables.

Example: '1 + α X ENTER (you get '1+X'); '1 ÷ α X ENTER ('1/X'); y^x *Result:* '(1+X)^(1/X)'

To calculate with such expressions, use → SYMBOLIC. It offers you a choice of six different calculation tools, with a screen for each.

Solving a Quadratic Equation

Example: Solve symbolically for the (principal) solution of the quadratic equation $5x^2 - 11x + 2 = 0$. Press → SYMBOLIC, use ▼ to highlight Solve quad..., and press OK. In the EXPR: field (the arrow keys will move the highlight around), use the EquationWriter or the command line to enter 5*X^2-11*X+2. In the VAR: field, enter X (*uppercase*—it must match your expression). In the RESULT: field, put Symbolic (use the CHOOS box and ▼ as necessary). Select PRINCIPAL solution by putting a √ (via ☑ CHK). Now you're ready to solve, so press OK.... 'X=2'

When you get your result, you also get the ← SYMBOLIC menu, which offers other tools for altering, expanding or using an algebraic expression (see the Command Index, pages 41-77, for individual details):

← SYMBOLIC **(93)**

COLCT	EXPA	ISOL	QUAD	SHOW	TAYLR
↑MAT	↓MAT	→Q	→Qπ	I	APPLY
QUOT					

Now repeat the above quadratic sample but request the *general* solution.... The ±1 variable appears in the solution as the ± in the quadratic formula. To get either of the two specific solutions, you store a −1 or 1 into the name ±1, then request a numeric solution.*

The other five tools in → SYMBOLIC have similar input screens—note the self-explanatory prompts for each field. More examples follow on the next 4 pages.--->

*You're actually altering the states of Flags -1 and -3 via the RESULT: and PRINCIPAL: items on the quadratic input screen. Always bear in mind that the states of Flags -1, -2 and -3 can affect the evaluation of a symbolic expression—see page 78.

Isolating a Variable*

Example: To solve $\sin x = 0$: Press [→][SYMBOLIC], then select `Isolate var...`. Enter the expression into `EXPR:` ['][SIN][α][X][ENTER]. In `VAR:`, specify the variable to be isolated: [α][←][X][ENTER]. Then, with `_PRINCIPAL` *un*checked—to get the general solution—press **OK**. Result : `'X=180*n1'` (*or* `'X=π*n1`). Then `n1` is a parameter representing an arbitrary integer; any integer multiple of 180° (π radians) is a solution.

Computing Taylor Polynomials

Example: Find the 5th-degree Taylor polynomial of $\sin x$ about $x = 0$. Press [←][RAD][→][SYMBOLIC], then select `Taylor Poly...`. Enter the expression into `EXPR:` ['][SIN][α][X][ENTER]; in `VAR:`, specify the variable to be isolated: [α][X][ENTER]. In the `ORDER:` field, enter [5]. Be sure that the `RESULT:` field says `Symbolic` and press **OK**. *Result:* `'X-1/3!*X^3+1/5!*X^5'` (Press [▼] to see it more clearly in the EquationWriter.)

The `Taylor Poly...` tool can calculate expansions only about $x = 0$, but you can also use it for calculations about $x = a$ by using *variable substitution:* You re-express your function in terms of another variable—say, `W`—where $x = w + a$. Then calculate the expansion and re-substitute ($w = x - a$). For simpler functions, you can do the initial substitution by inspection; for more involved expressions, it's easier to let the machine help you: Just store the substitution `'W+a'` (you choose the numerical value of a here) into the variable `X`. Be sure that the variable `W` does not exist in the current directory path.

Example: Find a 4th-degree Taylor expansion of $\ln x$ about $x = 1$. By inspection, you can replace $\ln x$ with $\ln(w + 1)$. Now find its expansion around $w = 0$: Press [→][SYMBOLIC] and select `Taylor Poly...`. Now put `'LN(W+1)'` in the `EXPR:` field, `W` in `VAR:` and `4` in `ORDER:`, and press **OK**. (*Result:* `'W-.5*W^2+2/3!*W^3-6/4!*W^4'`) Now make the re-substitution: ['][α][X][−][1][ENTER]['][α][W][STO]. [EVAL] the expansion. (And now you may want to use the rationalizing tool, →`Q`, to simplify the decimals into fractions—press [←][SYMBOLIC][NXT] **→Q**). Again, [▼] shows the result more clearly (use [▶] and [◀] to scroll as necessary).

*`Isolate var...` works only when the desired variable appears just *once* in the expression (any level of order); `Solve quad...` works for a variable with multiple occurrences but *no higher than second order.*

∂ AND DIFFERENTIATION

To take a derivative, use Differentiate... (under →SYMBOLIC) or the command ∂ (via →∂).

Numerical Differentiation

Example: Find the slope of $1/x$ at $x = 2.7183$. Press →SYMBOLIC, then select Differentiate.... Enter the EXPR: '1÷αX ENTER. Specify the variable in VAR: αX ENTER. Set RESULT: to Numeric (use +/− if needed); and in the VALUE: field, enter 2·7183. Now press OK. *Result:* -.1353....

∂ can do the same thing on the stack: First, enter the expression ('1/X' here), then the variable, ('X' here). And for a numeric derivative, store the value in X: 2·7183 ENTER 'αX STO. Press →∂.

Symbolic Differentiation

Example: Find the slope of $1/x$ at any point. First for a symbolic derivative, you *purge* the variable name—'X' here (and you must do this throughout the current path): 'αX←PURG. Then proceed: Press →SYMBOLIC and select Differentiate.... Enter the EXPR: '1÷αX ENTER. Put the variable in VAR: αX ENTER. Set RESULT: to Symbolic (use +/− if needed). Press OK. *Result:* '-(1/X^2)'

∂ does the same thing on the stack: Enter the expression ('1/X'), the variable, ('X'), and press →∂.

Stepwise Differentiation

Example: Find the slope of $(\sin x)/x$ at any point.

First, for a symbolic derivative, you must *purge* the variable name, 'X' ('αX←PURG)—and you must do this throughout the current path. Next, set RAD mode (if it's not set already). Now press →SYMBOLIC and select Differentiate.... Next, enter the EXPR: 'SINαX▶÷αX ENTER; then put the variable of differentiation (X) into VAR: αX ENTER. Finally, set RESULT: to Symbolic (use +/− if necessary). Now, at this point, instead of pressing OK (which would do the complete differentiation), press STEP.... *Result:* '∂X(SIN(X))/X-SIN(X)*∂X(X)/X^2'

Now just press EVAL repeatedly to see each step in the differentiation.

Implicit Differentiation

When unable to express y explicitly as a function of x, try implicit means. *Example:* If $x^2 + y^2 = 9$, find dy/dx. First, purge 'X' and 'Y' from the current path. Next, enter the equation—*so that it says* "y *is a function of* x:" 'X^2+Y(X)^2=9'—then the variable, 'X', and →∂. *Result:* '2*X+derY(X,1)*2*Y(X)=0'. The expression derY(X,1) means dy/dx, which you now isolate: Edit the equation, replacing derY(X,1) with, say, DY (and Y(X) with simply Y). After substitution: '2*X+DY*2*Y=0'. Solve for DY: 'αDαY ENTER ←SYMBOLIC ISOL COLCT. *Result:* 'DY=-(X/Y)'

Partial Differentiation

The HP 48G/GX can take partial derivatives of multi-variable functions. All variables except the variable of differentiation are treated as constants (if you don't express them as being functions of the variable of differentiation—unlike implicit differentiation, above).

∫ AND INTEGRATION

To compute an integral, you can use the ∫ command (via →∫), or Integrate... (under →SYMBOLIC).

Numeric Integration

Example: Find $\int_0^1 \frac{1}{1+t^2}\,dt$. Try the EquationWriter, which is very handy for integrals: Press ←EQUATION →∫0▶1▶1÷1+αTy^x2▶▶▶αTENTER. This puts '∫(0,1,1/(1+T^2),T)', the algebraic form, onto the stack. Now press EVAL once to see the integral before substituting numeric limits:

```
'1*(ATAN(T)/∂T(T))|(T=1)
        -(1*(ATAN(T)/∂T(T))|(T=0))'
```

Use EVAL again until you get a numeric result (≈.7854). This is an exact integral, an explicit antiderivative, but sometimes EVAL leaves an expression still containing an integral (e.g. '∫(0,1,e^-(X^2),X)'); the machine can't find an antiderivative. It knows antiderivatives for most built-in functions (and their derivatives), and it automatically does simple linear substitutions, but not much more. However, you can still get a good result via numerical estimation....

Example: Repeat the above example, but use ◀ ▶NUM instead of EVAL. Note the appearance of **IERR** in the VAR menu. The number in IERR is a likely bound on the maximum error in the estimated integral; the accuracy is set by the display mode: More digits offer more accuracy (and increased calculation time). Compare results with a display set to STD, then FIX 2.

To evaluate a ***multiple integral***, key it in just as you would write it—one integral nested inside the other—and, again, the EquationWriter is handy for entering.

Example: $\int_0^4 \int_1^9 2xy\, dy\, dx$

Then use ◀ ▶NUM or EVAL as usual (though you may need more repetitions of EVAL (or even EXPAND and COLCT) to get a final explicit result if indeed possible.

To evaluate an ***improper integral*** (where the limits may be infinite), *change variables*. If x is unbounded, one handy transformation is $w = \tan^{-1}x$. Then $\tan^{-1}(\pm\infty) = \pm\pi/2$, and $f(x)dx = f(\tan w)(1+\tan^2 w)dw$.

Symbolic Integration

Example: Integrate $\int_1^x \frac{1}{t}\, dt$. First, purge 'T' and 'X' from the current path, then use ↱SYMBOLIC and select Integrate... Put '1/T' in EXPR:, T in VAR:, 1 in LO: and X in HI:. Set RESULT: to Symbolic, and **OK**. *Result:* '1*(LN(T)/∂T(T))|(T=X)-(1*(LN(T)/∂T(T))|(T=1))'. Press EVAL: 'LN(X)'

Example: Find the antiderivative of $3(x+5)^2$. Purge 'X' from the current path, then use ↱SYMBOLIC and select Integrate... Enter '3*(X+5)^2' in EXPR:, X in VAR:, 0 in LO: and X in HI:. Put Symbolic into the RESULT: field, and press **OK**. *Result:*

'3*((X+5)^(2+1)/((2+1)*∂X(X+5)))|(X=X)-(3*((X+5)^(2+1)/((2+1)*∂X(X+5)))|(X=0))'

Now, don't press EVAL yet. First, discard the second part of this result—that for the zero lower limit: PRG **TYPE** **OBJ▸** DROP DROP DROP. *Now* press EVAL. *Result:* '3*((X+5)^3/3)' (You can further simplify this via **EXPA** and **COLCT** from ◀ SYMBOLIC.)

These examples work because their integrands have easy antiderivatives. For an integrand that doesn't, you can get a *symbolic approximation*: Integrate the Taylor polynomial approximation of the integrand.

MATRIX, STAT AND Σ

Although its name seems so similar to the statistical commands (such as ΣDAT, Σ+ and Σ−), the Σ command (available via →Σ) is *not* directly tied to statistics; you can use Σ to sum any finite series.

Example: Sum the first k powers of 2 ($2^0 + 2^1 \ldots + 2^k$): Enter 'N', then 0, then 'K', then '2^N', and →Σ; *or:* Enter 'Σ(N=0, K, 2^N)'—and you can use the EquationWriter to do this. Any way you do it, it means "Find the sum of 2^n for all integers n from 0 to k."*

Contrast that with the true statistics functions, which all relate to the matrix (array) of data currently stored in the reserved name ΣDAT. Recall that an array can contain either real or complex values (see page 1), and that you can enter an array on the stack directly via the [[ ]] syntax.** But there's also a special tool for easy matrix entry and editing: →MATRIX

Example: Enter the following test score matrix and name it TESTS:

64	80	97	88
79	53	67	75
93	88	90	93
55	64	70	77
99	95	82	96
70	62	65	74
89	83	87	85

Press →MATRIX to start the MatrixWriter. Its menu lets you choose how wide each column is (via ◆WID and WID▶) and how the cursor moves after each entry (either across, with GO▶■; down, with GO▼■; or nowhere, with GO▶ and GO▼).*** For this example, set STD notation, leave the column width as is, and enter data by row (i.e. use GO▶■): 64 ENTER 80 ENTER 97 ENTER 88 ENTER, then ▼ (this establishes the column count in the matrix; from now on, the machine will know when to begin a new row, so you can finish data entry uninterrupted); 79 ENTER 53 ENTER 67 ENTER 75 ENTER, etc. An extra ENTER exits the MatrixWriter and puts your completed matrix onto the stack—just as if you had keyed it in there. Now name it: ' α α TESTS ENTER STO.

*Note that the nature of your result (symbolic vs. numeric) will depend on the states of Flags -2 and -3. See page 78 for more about those flags.

**A vector is also a kind of array—though it has its own syntax, [ ]; depending on the circumstances, it can be treated by the machine as being either a row matrix or column matrix.

***And the 2nd page of the menu has tools to add/extract rows/columns.

Now suppose your data represent exam scores—3 midterms and a final exam—for each of 7 students.

Example: Compare the mean and (population) standard deviation of the first exam (column 1) with those of the final exam (col. 4). Press → STAT and select Single-var... At the ΣDAT: field, use CHOOS to indicate the matrix named TESTS. Then set COL: to 1, TYPE: to Population, ☑ CHK the _MEAN and _STD DEV fields, and press OK. Tagged results go onto the stack (Mean: 78.42...; Std Dev: 15.00...). Just repeat this process (but use col. 4) to compare.

Notice the other calculation options you have under Single-var... statistics (_TOTAL, _MAXIMUM, etc.). Also, explore the other handy → STAT tools:

Frequencies... lets you study data distribution by defining *bins*—intervals of value—then calculating how many data points fall into each bin.

Fit data... examines correlations between any 2 designated columns of data in ΣDAT. You can try linear, logarithmic, exponential or power-curve models.

Summary stats... computes the often-used quantities Σx, Σy, Σx^2, Σy^2, Σxy and Σn (for any 2 columns you designate as x and y in ΣDAT).

Note that, although most of the matrix-building and *mathematical* matrix-crunching commands are offered via MTH, the matrix *statistical* commands are under ← STAT (see the Command Index, pp. 41-77, for individual details on these commands):

← STAT (96)

(97) DATA
Σ+ | Σ- | CLΣ | ΣDAT | | STAT

(98) ΣPAR
XCOL | YCOL | MODL | ΣPAR | RESET | INFO
| | | | | STAT

(99) MODL
LINFI | LOGFI | EXPFI | PWRFI | BESTF | ΣPAR

(100) 1VAR
TOT | MEAN | SDEV | MAXΣ | MINΣ | BINS
VAR | PSDEV | PVAR | | | STAT

(101) PLOT
BARPL | HISTP | SCATR | | | STAT

(102) FIT
ΣLINE | LR | PREDX | PREDY | CORR | COV
PCOV | | | | | STAT

(103) SUMS
ΣX | ΣY | ΣX^2 | ΣY^2 | ΣXY | NΣ
| | | | | STAT

PRG (22)

(23) BRCH
IF | CASE | START | FOR | DO | WHILE
IFT | IFTE

(24) IF
IF | THEN | ELSE | END | BRCH

(25) CASE
CASE | THEN | END | BRCH

(26) START
START | NEXT | STEP | BRCH

(27) FOR
FOR | NEXT | STEP | BRCH

(29) DO
DO | UNTIL | END | BRCH

(31) WHILE
WHILE | REPEA | END | BRCH

(32) TEST
== | ≠ | < | > | ≤ | ≥
AND | OR | XOR | NOT | SAME | TYPE
SF | CF | FS? | FC? | FS?C | FC?C
LININ

(33) TYPE
OBJ→ | →ARR | →LIST | →STR | →TAG | →UNIT
C→R | R→C | NUM | CHR | DTAG | EQ→
TYPE | VTYPE

(34) LIST
ELEM | PROC | OBJ→ | →LIST | SUB | REPL

(35) ELEM
GET | GETI | PUT | PUTI | SIZE | POS
HEAD | TAIL | LIST

(36) PROC
DOLIS | DOSUB | NSUB | ENDS | STREA | REVLI
SORT | SEQ | LIST

(37) GROB
→GRO | BLAN | GOR | GXOR | SUB | REPL
→LCD | LCD→ | SIZE | ANIM

(38) PICT
PICT | PDIM | LINE | TLINE | BOX | ARC
PIXON | PIXOF | PIX? | PVIEW | PX→C | C→PX

(39) IN
INFOR | NOVA | CHOOS | INPUT | KEY | WAIT
PROM

(40) OUT
PVIEW | TEXT | CLLCD | DISP | FREE2 | MSGB
BEEP

(41) RUN
DBUG | SST | SST↓ | NEXT | HALT | KILL
OFF

(61) ERROR
DOERR | ERRN | ERRM | ERRO | LASTA | IFERR

(60) IFERR
IFERR | THEN | ELSE | END | ERROR

You'll find most of the commonly used postfix programming commands collected in groups under the PRG key. Recall that all postfix commands consume their arguments before leaving their stack results (if any). Here is an overview of the various groups, with some highlights noted (for more details on a command, see the Command Index on pages 41-77):

The program ***branching*** commands (under **BRCH**) let you build programs that make execution decisions "on the fly," testing values or conditions and directing program control according to the results of those tests via constructs such as IF-THEN-ELSE, START-STEP, FOR-NEXT, DO-UNTIL, WHILE-REPEAT, and CASE statements. *Notes:* The test that triggers a branching decision may be of any length; the sole issue is whether its ultimate result is zero (false) or non-zero (true). For program repetitions ("loops"), use START or FOR if you know the number of repetitions; use DO or WHILE if you don't.

The ***test*** commands (under **TEST**) test the states of flags or compare values, allowing decisions (such as program branching—see above) during program execution. *Notes:* The logical tests (AND, OR, XOR and NOT) let you form compound tests by combining what would otherwise be separate tests. Two of the flag tests (FS?C and FC?C) clear the given flag after testing its status. The two equality tests, == and SAME, are similar except when testing algebraic objects.

The ***type*** commands (under **TYPE**) offer commands that form various object types from their constituent parts. *Notes:* As for the converse, the one necessary command—an all-purpose object *de*-constructor (for most object types)—is OBJ→. Also, two commands (TYPE and VTYPE) test the type of a given object (see also page 1).

The ***list*** commands (under **LIST**) offer two groups of commands. Those in the first group manipulate just a single given element in a given list; those in the other group run procedures on all elements in a given list. *Notes:* Various combinations of commands such as HEAD, TAIL, REVLIST and SORT offer powerful ways to quickly re-order and present lists (and recall that you can put almost any type of object into a list). And DOLIST, DOSUB, STREAM and SEQ will quickly send each element in turn through a given process, with the result(s) appearing in a list, too.

The ***grob*** commands (under GROB) perform a wide variety of manipulations on any given *grob* (graphics object). *Notes:* The two LCD commands let you easily save and recall screen displays. And the ANIMATE command allows you to define a set of fixed grobs, then cycle the display through them quickly enough to achieve the effects of animation.

The ***PICT*** commands (under PICT) perform a wide variety of manipulations on the grob currently stored in the reserved name PICT. *Notes:* Keep in mind that you can specify the pixels in a grob either in *pixel coordinates*, via a list of binary integers, { #*horizontal* #*vertical* }; or in user-defined coordinates, via a complex number, (*x*, *y*). The scale of the user-defined units depends on the current settings stored in PPAR. There are also handy commands for quickly drawing lines, boxes and arcs.

The ***input*** commands (under IN) give you ways to let a user input values during program execution. *Notes:* You can use PROMPT or INPUT to cause a program to halt with a message asking for user response; or you can use WAIT or KEY to detect or identify user keystrokes; or you can use CHOOSE to build a "CHOOS-box" for the user to select from; or the INFORM command lets you build an *input screen*—similar to those of the built-in applications.

The ***output*** commands (under OUT) give you ways to format program output. *Notes:* You can sound the tone generator with BEEP; or your can use MSGBOX to convey intermediate messages during program execution; or you can use FREEZE or DISP to halt program execution to allow a user to inspect output.

The ***run control*** commands (under RUN) let you troubleshoot a program: With the name of the desired program as the stack argument, you execute DBUG, which initiates execution but then halts immediately. Thereafter, you can *single-step* (SST) through the execution to examine the effect of each program step (or SST↓ shows every subroutine step as well); or NEXT lets you look a couple of step ahead without executing those steps.

The ***error*** commands (under ERROR) offer ways to detect and recover from run-time errors. The IFERR-THEN-ELSE construct lets you execute a command *tentatively*—you can undo it if it causes an error.

TIME

→TIME lets you control the calculator's ***clock*** and set ***alarms*** for appointments or other time-delayed events.

To Set the System Clock: Press →TIME, highlight Set time, date... and press OK. Fill in each field as it is highlighted (move the highlight, as needed). For AM/PM (and for the date format) use CHOOS. If finished, OK accepts; CANCEL aborts.

There are two kinds of alarms—***appointment*** alarms and ***control*** alarms. An appointment alarm will display a (text) message and beep at you for about 15 seconds; a control alarm will simply evaluate a specified object, such as a program.

Setting Alarms: To set an alarm, press →TIME and select Set alarm.... Then, in the MESSAGE: field, for an appointment alarm, enter a string (e.g. "Phone home"); or, for a control alarm, enter the object you want evaluated at the specified time. Specify time, date and repeat interval,* then press OK.

Acknowledging Alarms: You must acknowledge appointment alarms, but not control alarms. If you catch an appointment alarm while it's beeping, press any key to acknowledge it. If its beeping has stopped (the (•) will show it as *past-due*), press →TIME for the message, then ACK to acknowledge it. (To acknowledge more than one such alarm, use ACKA.)

→TIME also offers menu access to other clock and alarm commands, plus commands for time arithmetic.** See the Command Index (pp. 41-77) for details:

→TIME (94)

DATE	→DAT	TIME	→TIM	TICKS	ALRM
DATE+	DDAYS	→HMS	HMS→	HMS+	HMS-
TSTR	CLKA				

(95) ALRM

ACK	ACKA	STOAL	RCLAL	DELAL	FINDA
					TIME

Note: Alarm behavior is affected by Flags -43, -44, and -57. See pages 78-80 for more details.

*Don't set an interval too brief to acknowledge the alarm before it repeats. (If this happens, use the ON-4 system interrupt—see p. 39.)

**Note, too, that the various "HMS" commands are useful for *angular degree* measure, as well as chronological time measure.

I/O AND DATA COMMUNICATIONS

The HP 48G/GX can transfer information to and from a variety of other devices. The thumbnail "recipes:"

HP 48G/GX <----> HP 48G/GX

Receiver: Move to the directory where the incoming data is to be stored and press → I/O. Then select Get from HP 48G/GX and press OK.

Sender: Press → I/O. Select Send to HP 48G/GX... and OK. At the NAME: field, press CHOOS and select the object(s) to be sent, then OK and SEND.

HP 48G/GX ----> Infrared Printer

Setting Up: Aim the calculator's IR ▲ at the printer's window (from a distance ≤ 18"). Clear Flag -34, then press ← I/O PRINT PRTPA RESET.

Printing: → I/O ▼ ▼ ▼ OK, then, if needed, change PORT: to Infrared (use +/−). Adjust _DBL-SPACE, DELAY:, and _LINEF, if needed. Now, either exit to use a ← I/O menu print command (or the ON-1 screen dump), or, at OBJECT:, use CHOOS to select object(s) to be printed, then OK and PRINT.

HP 48G/GX ----> Serial Printer

Setting Up: Connect the 9-pin end of the cable to the printer; the 4-pin end to calculator. If using XON/XOFF: ← I/O NXT SERIA OPENI NXT I/O CLOSE VAR → IOPAR ← EDIT, then change the 4th list element to 1. If the end-of-line sequence is not CR-LF, press ← I/O PR1, then edit the 4th list element.

Printing: → I/O ▼ ▼ ▼ OK. Set PORT: to Wire (use +/−) and adjust _DBL-SPACE, XLAT:, _LINEF, BAUD:, PARITY:, and LEN:, as desired. Now, either exit to use a ← I/O menu print command (or the ON-1 screen dump), or at OBJECT:, use CHOOS to select the object(s) to be printed, then just OK and PRINT.

HP 48G/GX <----> Computer via Kermit

Setting Up: Connect the 25-pin end of the cable (w/ adapter) to the computer and the 4-pin end to the calculator. On the computer, move to the desired directory, run the program containing Kermit and

make it the server (via a command such as SERVER). On the calculator, press → I/O ▲ ▲ OK. If needed, change PORT: to Wire (use +/−), TYPE: to Kermit; adjust other parameters to match your computer. (Binary transfer is faster, but use ASCII if you want to be able to edit transferred objects on the computer.)

Transferring: Press CHOOS (then ▼ if you're going to transfer from the computer), then OK to browse through possible object(s) to be transferred (put a check mark beside each one selected). Press OK to finalize the list of names. Now, to send the names *to* the computer, press SEND; or, to get the names *from* the computer, press SRVR KGET. Then (for either direction), press (SRVR, if necessary) FINIS.

Sending Backups Directly to the Computer: **Turn off the ticking clock display in the calculator**. Set up the transfer as usual (Binary is faster). On the calculator, put :IO: *name* onto the stack, where *name* is the file to be created on the computer. Then press ← MEMORY NXT ARCHI. When the session is finished, press ← I/O SRVR FINIS NXT CLOSE.

HP 48G/GX <----> Computer via other methods

The HP 48G/GX can transfer to a computer via other serial protocols, too (XModem is on the → I/O screen). For specific details, consult your computer's and/or main HP manuals. Note also that the product known as the *HP Connectivity Kit* (available for DOS/Windows or Macintosh), allows for especially simple data transfers/handling. Refer to its supplied instructions.

← I/O (104)

SRVR IOPAR PRINT PR1 SEND RECV
SERIA KERR RECN CLOSE XSEN XRECV

(105) SRVR
SERVE PKT KGET FINIS I/O

(106) IOPAR
IR ASCII BAUD PARIT CKSM TRAN
IOPAR RESET INFO I/O

(107) PRINT
PRVAR PRST PRSTC PRLCD CR PRTPA
PR1 I/O

(108) PRTPA
DELAY OLDPR PRTPA RESET INFO PRINT

(109) SERIA
XMIT SRECV STIME SBRK BUFLE OPENI
I/O

LIBRARY, PORTS AND BACKUPS

Port memory is different than user memory (where you normally store variables), in that port memory cannot be subdivided into directories, nor can you edit or view objects stored there. It is entirely *independent* memory—even when everything in main memory is purged, port memory is maintained—but it can contain just 2 object types: *Backup objects* and *Libraries*.

Port memory may be located in Port 0, Port 1, and/or Port 2, denoted by a (numerical) port identifier, :*port*: . *Port 0 is internal to the machine*; it is taken out of user memory, so that any space used as Port 0 reduces the amount of user memory then available. Ports 1 and 2 exist only in the HP 48GX—if there are RAM memory cards plugged into those ports.*

Backup Objects

A backup object is named with its port location as well as a normal name, in the format :*port*:*name*. ***To store any object into a backup object***, use STO with a backup identifier. *Example:* To store "Red" as a backup object named Color in Port 0, enter "Red" onto the stack, then key in :0:Color and press STO.

To get a menu of all backup objects currently stored in a given port, press ←LIBRARY PORTS, and then select the desired port (for example, press :0: now). A port menu behaves very much like the VAR menu: ***To evaluate*** a backup object, you can either key in its identifier (e.g. :0:Color) and press EVAL, or just press its menu key (e.g. COLOR). Likewise, ***to recall*** a backup object, either key in its identifier and use RCL or press → and its menu key (e.g. →COLOR). And ***to purge*** a backup object, put its identifier (or a list of such names) on the stack and use ←PURG.

To backup all user memory (the entire contents of HOME directory), *first be sure the ticking clock is not in the display*. Then key in a backup identifier (of your choice) and use the ARCHIVE command. To ***restore*** this information (*which replaces all current user memory*), enter its backup identifier and use RESTORE.

*Note that any RAM in card slot 1—where a card is limited to 128 Kb—may either be *merged* with normal user memory or left as port memory (denoted Port 1). But any RAM in card slot 2—where a card is limited to 4 Mb—must be left as port memory and is partitioned into ports of 128Kb each (i.e. Ports numbered 2–n, where $n \leq 33$).

Libraries

A library is a "read-only directory" of objects that you *attach* to a specific directory in your user memory, thereby making those objects "visible" (i.e. their names invocable) as part of that directory. Each library is identified by a unique number in port memory, in the format **:** *port***:** *number*. The number appears in a menu if you press ◀LIBRARY **PORTS**. However, to make it easier to identify during use, a library's *name* appears in a menu if you press ▶LIBRARY while in the attaching directory (or any subdirectory of it).

You can attach only one library at a time to any given directory (except **HOME**, which may have any number of attachments), but you can attach the same library to more than one directory at a time. However, before any library can become attached, it must first reside in port memory—either Port 0 internally or in one of the plug-in card slots. With a plug-in card, inserting it correctly into the required slot effectively places it in port memory (*but be sure to follow instructions carefully for plug-in and power-up*).

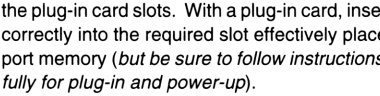
To place a library into Port 0, put the library object onto the stack, then key in the desired library identifier (**:** *port***:** *number*) and press STO—exactly analogous to naming and storing a backup object. (And purging a library is also analogous.)

To attach an auto-attaching library, simply turn the machine off, then back on. All auto-attaching libraries will then be attached to the **HOME** directory.

To attach (or detach) a library, move to the desired directory level, then key in the desired library identifier and use the ATTACH (or DETACH) command.

Other related commands, such as those for merging card-based RAM, appear via ◀LIBRARY. *Follow the card's operating instructions carefully* (and see pp. 41-77 here for more details on any single command):

◀LIBRARY **(110)**



EQ LIB AND THE MES

The Equation Library application (→ EQ LIB) is exactly that—an access tool that lets you browse through the built-in equation library, examining any equation group's formulas, variables, units, help diagrams, etc. And with the touch of a key, you can load the equation group into the ***Multiple Equation Solver*** (MES).

Example: If you were to stand and fire a rifle (muzzle velocity 500 m/s) horizontally from shoulder height (1.5 m) along a deserted stretch of highway, how far would it have travelled along the road in that time?

Press → EQ LIB and be sure that the menu items **UNIT** and **SI** appear with the little ■'s in them, indicating that the equations will indeed include units and that they will be SI units. Now use ▼ to highlight the topic Motion, and press ENTER. You'll see a selection of the various groups of motion equations.

Note how the menu items let you examine aspects of a group. Highlight Projectile Motion and press **PIC** for a picture.... Try **VARS** to see all the variables involved.... Press **EQN** to see the first equation in this group (you need to be patient as the EquationWriter presents it); use **NXEQ** to see each other in its turn. *At any point,* press **SOLV** and *all* equations in this group are loaded into the MES.

Now key in the known values: 0 **X0** 1 · 5 **V0** 0 **Y** 0 **A0** NXT 5 0 0 **V0**. And solve for the unknown: NXT NXT ← **X**.... X: 276.5_m. *The MES is smart enough to determine which equation(s) to use, based on the known value(s) you input.*

Notice that in the MES—just as in the ← SOLVE menu (see page 15)—to *store* a value, you key in the value and press its menu key; to *recall* a value, you press → and the menu key; to *calculate* a value, you press ← and the menu key. But here in the MES, any ***known*** value—any value you key in—goes ***dark*** (e.g. **V0** becomes **V0**) to indicate its known status. And then, when solving for an unknown, the MES denotes with ■'s the values it has used (and displays the intermediate calculations it performs in the process).

One other handy MES key: To reset all values to unknown status, press **ALL**; to calculate all unknowns, press ← **ALL**; to recall all values, press → **ALL**.

Of course, you can also use the MES on your own groups of equations—just put your desired equations into a *list.* And you can use the built-in group of projectile motion equations as the starting point.

Example: Suppose you want to know the trajectory angle(s), θ_0, needed for a target at range 6000 m, elevation 1200 m, with a 300 m/s muzzle velocity. You can't do this easily with just the built-in equations list, but you can add the necessary equation to a copy of that list, then store this as, say, PROJ:

Highlight Projectile Motion in the Equation Library (as in the previous example), but then just press +STK to send that group's equation list to the stack. Now just key in the necessary new equation:

```
'θ0=ATAN((y-y0)/(x-x0)+CONST(g)/2
          *((x-x0)/(v0^2*COS(θ0)^2)))'
```

ENTER this onto the stack and press + to add it to the list already there. Name this, '[α][α]PROJ ENTER STO, and it's ready to use in the MES—via ↶ EQ LIB:

↶ EQ LIB **(113)**

(114) EQLIB

EQNLI	SOLVE	MSOL			EQLIB

(115) CDLIB

CONLI	CONS				EQLIB

(116) MES

MSOL	MINIT	MITM	MUSE	MCAL	MROO
					EQLIB

(117) UTILS

MINE	2FACT	FANNI	DARCY	FOX	SIDEN
TOELT	TINC	GMOL	LBMO	RPM	OE
					EQLIB

Press VAR PROJ '[α][α]EQ[α]STO, to denote your list as the current equation set. Now press ↶ EQ LIB MES, initialize (MINIT), and start the MES (MSOL). If you use the data above, you'll get a trajectory angle of about 34.07° or 67.23°, depending on your initial guess for θ0 (don't forget about multiple solutions—guessing works just as in the regular Solver).*

See the Command Index (pages 41-77) for details on the other items in ↶ EQ LIB—more MES commands, the Constants Library, the MINEHUNT game, etc.

*Note, too, that the presence of units in your own MES equation lists is not automatic. As in the regular Solver, if the variables do not already exist, they will be created by the MES without associated units; you must then key in the desired units with your initial input values.

CST AND CUSTOM MENUS

Pressing CST is the same as executing 1 MENU (see page 10 for more on menus). This ***creates a custom menu*** from a ***list*** you've stored in the reserved name CST. That is, before using CST, you must have already entered such a list, then entered 'CST', and pressed STO. The menu list is of this form:

```
{  {  "menu label_A" {  unshifted object_A
                        ←-shifted object_A
                        →-shifted function_A
                     }
   }
   {  "menu label_B" {  unshifted object_B
                        ←-shifted object_B
                        →-shifted object_B
                     }
   }  ... (etc.)
}
```

(You may omit the innermost bracketing if there are objects only for the unshifted menu keys; you may omit the labels and their bracketing if you don't wish the menu items to have labels different than their corresponding unshifted objects.)

You can also make a ***temporary*** custom menu: Enter a menu list (just as for CST), but don't store it into CST. Instead, execute TMENU, which builds and displays a menu from *that* list—without referring to CST.

CUSTOM KEY ASSIGNMENTS

When the HP 48G/GX is in USER mode (←USER toggles much like α), its keys behave according to a ***list*** of objects mapped to keys via ***keycodes*** of the form *RowCol.Shift*. *Shift* values signify as follows: 0 or 1 = unshifted; 2 = ←-shifted; 3 = →-shifted; 4 = α-shifted; 5 = α←-shifted; 6 = α→-shifted. Thus the keycode for SPC is 94.0 or 94.1; the keycode for the K key (α←K) is 25.5.

To assign object(s) to key(s), enter the pair(s) in a list, { obj_A $kcode_A$ obj_B $kcode_B$ ... }, then use STOKEYS.

To un-assign keys (i.e. revert them to their standard functions), enter a list of the desired keycodes and use DELKEYS. To ***un-assign all keys***, use 0 DELKEYS.

To disable all un-assigned keys (i.e. assign them to "nothing"), enter 'S' and use DELKEYS. ***To then re-enable some keys*** to their standard functions, enter a list, { 'SKEY' $kcode_A$ 'SKEY' $kcode_B$ ... }, and use STOKEYS. ***To re-enable all keys***, use 0 DELKEYS.

To re-enable all other keys while assigning some, put S as the first object in your assignment list, then use STOKEYS as usual.

You can view the entire key assignments list at any time by using RCLKEYS.

System Information

Batteries

The HP 48G/GX uses 3 AAA batteries (lower compartment). Batteries usually last several months—and the (•) low-battery annunciator gives ample advance warning. The memory will also survive a few minutes without batteries during their replacement.

System Operations

To do system-level operations on your HP 48G/GX, you press *and hold down* the ON key, then press and release one or more other keys (then release ON):

ON-A-F	"Cold" restart (***erases all memory***).
ON-B	Cancels keystroke (prior to key release).
ON-C*	"Warm" restart (preserves memory).
ON-D	Starts interactive self-test.
ON-E	Starts continuous self-test.
ON-SPC	Deep-sleep shutdown (***turns off timer***).
ON-1	Performs display screen dump.
ON-4	Cancels next repeating alarm.
ON-−/ON-+	Adjusts display contrast.

Ports

The upper end of the HP 48G/GX has the 4-pronged ***serial port*** that connects to a Macintosh or IBM-compatible computer. Under the plastic end plate are two infrared "eyes" to allow wireless I/O between your machine and another calculator or printer. (The arrow on the top front helps align the infrared beam.) And the HP 48GX model offers two rear slots for plug-in memory, software, or other interface cards.

*If this fails, try inserting a straightened paper clip for about 1 second in the hole under the left rear rubber foot (which is removable).

Frequently Asked Questions

Q: *Why I am getting an error message?*
A: Check the argument types and order for the command you're trying to use (see pp. 41-77).

Q: *How do I adjust the display to be easier to read?*
A: Use ON-+ or ON-− (see p. 39).

Q: *How do I change the format or number of decimal places displayed by the calculator?*
A: Use ←MODES **FMT** (see pp. 11 and 41-77).

Q: *What does an* E *in a number mean?*
A: It's scientific notation (e.g. 6.02E23 = 6.02×10^{23}).

Q: *Why am I getting wrong results with trig functions?*
A: Check the angle mode (see ←MODES on p. 11). For example, if you see the annunciator **RAD** or **GRAD**, the machine is not using degrees.

Q: *Why don't I get* 0 *when I use* →NUM *on* 'SIN(π)'*?*
A: If you want the numerical identity that comes from the symbolic constant π, don't use →NUM (rather, use EVAL, with flag -2 clear). →NUM does its calculation on a 12-digit *numerical approximation* of π, which is 3.14159265359. (No machine can use the actual numerical value of π—it has an infinite number of digits). And $\sin 3.14159265359 \neq 0$. For similar reasons, 2 √x ←X² doesn't return 2.

Q: *Why can't I find the variable(s) I stored?*
A: You're probably in a different directory now than when you stored the variable(s). See pp. 6-9.

Q: *Why am I getting an* Undefined Name *error when using* →NUM *or* EVAL*?*
A: At least one name in the given expression is undefined in the current path. →NUM always demands a numerical result; EVAL demands it whenever flag -3 is set (see pp. 6-9 and 78).

Q: *Why doesn't* EVAL *give me a number?*
A: EVAL demands a numerical result only when flag -3 is set. Otherwise, names are not evaluated (except symbolic constants, which reduce to numerical values if flag -2 is set). So if you definitely want a number, use →NUM. (See also pp. 7 and 78.)

Q: *My calculator is "locking" up or behaving strangely. How can I check and/or correct this?*
A: Try restart procedures or system tests (see p. 39).

Q: *How much free memory does my calculator have?*
A: Use MEM (see p. 9) to find out: ◁ MEMORY **MEM**.

Q: *What does the (•) annunciator indicate?*
A: A low battery (see p. 39) or past-due alarm (p. 31).

Q: *My machine seems to gradually slow down. Why?*
A: It may need to clean up borrowed or fragmented sections of memory. Use ON-C (see p. 39) to do a cleanup. (**This clears the stack and** PICT.)

Q: *Why do I get mixed units in the Equation Library Solver even when I specify* **ENG** *or* **SI** *?*
A: The Solver initializes only variables that don't already exist in the current directory. For unit consistency, at the equation list in Equation Library, press **VARS** NXT **PURG** before selecting units.

Q: *How do turn off the* **HALT** *annunciator?*
A: Use KILL (see p. 54): PRG NXT **RUN** **KILL**.

Command Index

The following Command Index lists all *commands* (programmable objects) available in the HP 48G/GX. It does not list *operations*, (non-programmable keyboard procedures); it's a programmer's memory-jogger. This booklet does not attempt to teach programming. That topic is large enough to merit at least one full-length book (see the inside back cover).

Each command name is accompanied by a *brief* (nay, terse) description and its keyboard access, if any.* Then opposite each command is a stack representation of its inputs and outputs. Object types are represented by surrounding delimiters, descriptive phrases, and/or by variable naming conventions (i.e. x, y, and other italicized *names* indicate real values; n and m stand for integers, z is a complex number; etc.).

Not all possible object types are given—only the most common. Often you can also use names or algebraic expressions that evaluate to required types/values. Not all error responses or flag dependencies are given (see pp. 78-80 for descriptions of system flags).

*If you do extensive programming and need to study certain commands more thoroughly, you will find the *HP 48G Series Advanced User's Reference Manual* from Hewlett-Packard to be very helpful.

Name (keyboard access) **Description**

ABS (MTH VECTR ABS) Absolute value/magnitude.
ACK (TIME ALRM ACK) Acknowledge oldest past-due alarm.
ACKALL (TIME ALRM ACKA) Acknowledge all past-due alarms.
ACOS (ACOS) Inverse cosine.
ACOSH (MTH HYP ACOSH) Inverse hyperbolic cosine.
ADD (MTH LIST ADD) Add corresponding list elements (or add a single value to each element in a list).
ALOG (10^x) Common (base 10) antilogarithm.
AMORT (SOLVE TVM AMOR) Amortize a loan using values currently stored in the TVM variables (*I%YR*, *PV*, *PMT* and *PYR*), flag -14, and number of payments, *n*.
AND (PRG TEST NXT AND) Find logical AND of two arguments.
ANIMATE (PRG GROB NXT ANIM) Display a set of *n* graphics objects sequentially, in the area of the screen given by pixel coordinates *#X* and *#Y*, with a given time interval of delay, *int.* (specified in seconds), between each display.
APPLY (SYMBOLIC NXT APPLY) Create expression with given function name and arguments.
ARC (PRG PICT ARC) In the PICT grob, draw an arc with the given *radius*, centered at (x,y), going counterclockwise from arc position $angle_1$ to arc position $angle_2$.

ARCHIVE (MEMORY NXT ARCHI) Create a backup copy of HOME directory in independent RAM.
ARG (MTH NXT CMPL ARG) Complex number angle.
ARRY→ (must be typed) Decompose an array (either matrix or vector) into its constituent elements (in order) on the stack, together with description of the original dimensions of the argument array.
→ARRY (PRG TYPE →ARR) Build an array (matrix or vector) from constituent elements taken (in order) from the stack, using a description of the intended dimension(s) of the argument array.
ASIN (ASIN) Inverse sine.
ASINH (MTH HYP ASINH) Inverse hyperbolic sine.
ASN (MODES KEYS ASN) Assign the given object (or 'SKEY' un-assigns) to the key *rc.p* (row-col.plane).
ASR (MTH BASE NXT BIT ASR) Shift bits right, except left bit (adjacent bit becomes 0); right bit is lost.
ATAN (ATAN) Inverse tangent.
ATANH (MTH HYP ATAN) Inv. hyperbolic tangent.
ATICK (PLOT PPAR NXT ATICK) Put tick marks on axes at given interval(s)—in user units or in pixels.
ATTACH (LIBRARY NXT ATTAC) Attach given library.

Stack Inputs	→	Stack Outputs
1: *z* or [[*array*]]	→	1: |*z*| or ||*array*||
	→	
	→	
1: *z*	→	1: $\cos^{-1} z$
1: *z*	→	1: $\cosh^{-1} z$
2: { obj_a obj_b... }		1: { obj_a+obj_1
1: { obj_1 obj_2... }	→	obj_b+obj_2... }
1: *z*	→	1: 10^z
		3: *principal*
		2: *interest*
1: *n*	→	1: *balance*
2: *ufv*$_1$		
1: *ufv*$_2$	→	1: 1 or 0
n+1: $grob_n$		*n*+1: $grob_n$
:		:
3: $grob_2$		3: $grob_2$
2: $grob_1$		2: $grob_1$
1: { *n* { #*X*#*Y* } *int.* }	→	1: { *n* { #*X*#*Y* } *int.* }
2: { alg_1 alg_2 ... alg_n }		1: '*name*(alg_1 alg_2
1: '*name*'	→	...alg_n)'
4: (*x*, *y*) or { #*n* #*m* }		
3: *radius* or #n_{radius}		
2: *angle*$_1$		
1: *angle*$_2$	→	
1: :*port*:*name*	→	
1: (*x*, *y*)	→	1: *θ*
		nm+1: z_{11}
		:
		2: z_{nm}
1: [[*array*]]	→	1: { *n* } or { *n m* }
nm+1: z_{11}		
:		
2: z_{nm}		
1: *n* or { *n m* }	→	1: [[*array*]]
1: *z*	→	1: $\sin^{-1} z$
1: *z*	→	1: $\sinh^{-1} z$
2: *obj* or 'SKEY'		
1: *rc.p*	→	
1: #n_a	→	1: #n_b
1: *z*	→	1: $\tan^{-1} z$
1: *z*	→	1: $\tanh^{-1} z$
1: *x* or { *x*, *y* }	→	
1: *library number*	→	

Name (keyboard access)	Description
AUTO (◁PLOT NXT AUTO)	Autoscale display range(s).
AXES (◁PLOT PPAR NXT AXES)	Specify origin location, tick marks, and axis labels as part of PPAR.
BAR (◁PLOT NXT STAT PTYPE BAR)	Plot type=bar.
BARPLOT (◁STAT PLOT BARPL)	Plot bar chart.
BAUD (◁I/O IOPAR BAUD)	Set bit-transfer (*baud*) rate.
BEEP (PRG NXT OUT NXT BEEP)	Produce a tone of given *frequency* (in hz.) for given *duration* (in seconds).
BESTFIT (◁STAT ΣPAR MODL BESTF)	Find best-fitting Linear Regression model; note in PPAR.
BIN (MTH BASE BIN)	Select binary base mode.
BINS (◁STAT 1VAR BINS)	Sort elements of XCOL in the ΣDAT matrix into k+2 bins of specified width, with first such bin starting at data value x_{min}.
BLANK (PRG GROB BLAN)	Create blank grob of given height and width.
BOX (PRG PICT BOX)	In PICT, draw a box with given opposite corner locations (either pixels or user units).
BUFLEN (◁I/O NXT SERIA BUFL)	Count characters in serial buffer and check for data reception accuracy.
BYTES (◁MEMORY BYTES)	Determine byte count and checksum of given *object*.
B→R (MTH BASE B→R)	Convert binary to real.
CASE (PRG BRCH CASE CASE)	Begin CASE statement structure.
CEIL (MTH REAL NXT NXT CEIL)	Least integer ≥ x.
CENTR (◁PLOT PPAR NXT CENT)	Center plot on given point.
CF (◁MODES FLAG CF)	Clear the given *flag*.
CHOOSE (PRG NXT IN CHOOS)	Create user-defined choose box, using *message*, choice-result object pair list, initial highlight position; signal completion w/result.
%CH (MTH REAL %CH)	Calculate change from x to y, as a percentage of x.
CHR (◁CHARS CHR)	Return character of given code.
CKSM (◁I/O IOPAR CKSM)	Set error-detection *type*.
CLEAR (◁CLEAR)	Clear all objects from the stack.
CLKADJ (◁TIME NXT NXT CLKA)	Adjust system clock by x/8192 seconds.
CLLCD (PRG NXT OUT CLLCD)	Blank stack display.
CLOSEIO (◁I/O NXT CLOSE)	Close serial and IR ports; clear input buffer and KERRM error messages.
CLΣ (◁STAT DATA CLΣ)	Purge ΣDAT.
CLTEACH (must be typed)	Purge EXAMPLES subdirect.
CLUSR	Same as CLVAR.
CLVAR (must be typed)	Purge all variables and empty subdirectories in current directory.
CNRM (MTH MATR NORM %CH)	Array *column norm*.

Stack Inputs	→	Stack Outputs
1: { (x, y) *atick* "*x-axis label*" "*y-axis label*" } →		
1: *baud rate* →		
2: *frequency*		
1: *duration* →		
3: x_{min}		
2: *bin width*	2:	[[$n_{bin1} \ldots n_{bin\,k}$]]
1: *k* →	1:	[$n_{bin\,L}$ $n_{bin\,H}$]
2: # *width*		
1: # *height* →	1:	*grob*
2: (x_{UL}, y_{UL})		
1: (x_{LR}, y_{LR}) →		
	2:	*number of characters*
	→ 1:	0 or 1
	2:	# *checksum*
1: *obj* →	1:	*bytes*
1: # *n* →	1:	*n*
1: t/f →		
1: *x* →	1:	CEIL(*x*)
1: (x, y) →		
1: *flag* →		
3: "*message*"		
2: { { c_1 r_1 } ... { c_n r_n } }	2:	r_{choice}
1: pos_i →	1:	0 or 1
2: *y*		
1: *x* →	1:	100(*y*–*x*)/*x*
1: *n* →	1:	"$char_n$"
1: *checksum type* →		
1: *x* →		
1: [[*array*]] →	1:	*column norm*

Name (keyboard access)	Description

→COL (MTH MATR COL →COL) Decompose the given *array* (matrix or vector) into its constituent *columns* (which are vectors in a matrix and real or complex numbers in a vector); return also the *column count*.

COL+ (MTH MATR COL COL+) Insert one or more columns into the given *array* (an array into a matrix; an element or vector into a vector) at *position*$_1$.

COL- (MTH MATR COL COL+) Remove one column from the given *array* at *position*$_1$.

COL→ (MTH MATR COL COL→) Build an array (matrix or vector) from its constituent *columns* (which are vectors in a matrix and real or complex numbers in a vector), as indicated by the *column count*.

COLCT (↶ SYMBOLIC COLCT) Collect like terms in *expr*.

COLΣ (must be typed) Specify independent (x) and dependent (y) data columns in ΣDAT.

COMB (MTH NXT PROB COMB) Compute total number of possible combinations of n items taken k at a time.

CON (MTH MATR MAKE CON) Create an array of the given dimensions, filled with the same constant value, z. (The given dimensions may be an existing array, also.)

COND (MTH MATR NORM COND) Condition of *sq. mat.*

CONIC (↶ PLOT PTYPE CONIC) Set plot type to Conic.

CONJ (MTH NXT CMPL NXT CONJ) Conjugate a complex number (or complex array).

CONLIB (↶ EQ LIB COLIB CONLI) Open Constants Lib.

CONST (↶ EQ LIB COLIB CONS) Value of *constant*.

CONT (↶ CONT) Resume execution of halted program.

CONVERT (↶ UNITS CONV) Convert one unit object to the dimensions of another.

CORR (↶ STAT FIT CORR) *Correlation coefficient*.

COS (COS) Find cosine of z.

COSH (MTH HYP COSH) Find hyperbolic cosine of z.

COV (↶ STAT FIT COV) Find sample *covariance*.

CR (↶ I/O PRINT CR) Print contents of print buffer.

CRDIR (↶ MEMORY DIR CRDIR) Create directory.

CROSS (MTH VECTR CROSS) Compute cross product of two 2- or 3-element vectors.

CSWP (MTH MATR COL CSWP) Swap the specified *columns* in given *array* (*columns* are simply elements if given array is vector).

CYLIN (↶ MODES ANGL CYLIN) Set Cylindrical mode.

C→PX (MTH PICT NXT C→PX) User units → pixel units.

C→R (MTH NXT CMPL C→R) Separate real and imaginary portions of complex number (or array).

DARCY (↶ EQ LIB UTILS DARCY) Calculate Darcy friction factor for fluid flows.

DATE (↶ TIME DATE) System date (format: flag -42).

Stack Inputs	→	Stack Outputs
	n+1:	[*column*$_i$]
	:	
	2:	[*column*$_n$]
1: [[*array*]]	→ 1:	*column count*
3: [[*target array*]]		
2: [[*column(s)*]]		
1: *position*$_i$	→ 1:	[[*result array*]]
2: [[*target array*]]	2:	
1: *position*$_j$	→ 1:	[[*result array*]]
n+1: [*column*$_1$]		
:		
2: [*column*$_n$]		
1: *column count*	→ 1:	[[*array*]]
1: '*expr*'	→ 1:	'*simplified expr.*'
2: *col*$_x$		
1: *col*$_y$	→	
2: *n*		
1: *k*	→ 1:	C(*n*,*k*)
	1:	[[*z z ... z*]
2: { *n* } or { *n m* }		:
1: *z*	→	[*z z ... z*]]
1: [[*square matrix*]]	→ 1:	*condition number*
1: (*x*, *y*)	→ 1:	(*x*, -*y*)
1: '*name of constant*'	→ 1:	*value of constant*
2: *a_units*$_a$		
1: *b_units*$_b$	→ 1:	*c_units*$_b$
	→ 1:	*correlation coefficient*
1: *z*	→ 1:	cos *z*
1: *z*	→ 1:	cosh *z*
	→ 1:	*covariance*
1: *name*	→	
2: [*A*]		
1: [*B*]	→ 1:	[*A* **X** *B*]
3: [[*array*]]		
2: *column*$_i$		
1: *column*$_j$	→ 1:	[[*modified array*]]
1: (*x*, *y*)	→ 1:	{ #*n* #*m* }
	2:	*x*
1: (*x*, *y*)	→ 1:	*y*
2: *relative roughness*		
1: *Reynolds number*	→ 1:	*Darcy factor*
	→ 1:	*MM*.*DDYYYY*

Name (keyboard access)	Description
→DATE (◁ TIME →DAT)	Set system date (see flag -42).
DATE+ (◁ TIME NXT DATE+)	Calculate date which is *x* days from given date.
DDAYS (◁ TIME NXT DDAYS)	Calculate number of *days* between two given dates.
DEC (MTH BASE DEC)	Set decimal base mode.
DECR (◁ MEMORY ARITH DECR)	Reduce *value* by 1.
DEFINE (◁ DEF)	Store given *expr* into given *name*.
DEG (◁ MODES ANGL DEG)	Set Degree angle mode.
DELALARM (◁ TIME ALRM DELAL)	Delete given *alarm*.
DELAY (◁ I/O PRINT PRTPA DELAY)	Set time *delay* (in seconds) to be allowed after each line sent to printer.
DELKEYS (◁ MODES KEYS DELK)	Delete given key assignment(s); use 0 for all; 'S' blocks standard keys.
DEPND (◁ PLOT PPAR DEPN)	Set depend. variable.
DEPTH (◁ STACK DEPTH)	Number of objects on stack.
DET (MTH MATR NORM NXT DET)	Matr.*determinant*.
DETACH (◁ LIBRARY DETAC)	Detach specified *library*.
DIAG→ (MTH MATR NXT DIAG→)	Form *matrix* of given dim., w/elements from given array as major *diagonals*.
→DIAG (MTH MATR NXT →DIAG)	Inverse of DIAG→.
DIFFEQ (◁ PLOT PTYPE DIFFE)	Plot type = DIFFEQ.
DISP (PRG NXT OUT DISP)	Display *obj* on *n*th line of display ($1 \le n \le 7$; line 1 is top line).
DO (PRG BRCH DO DO)	Begin DO loop.
DOERR (PRG NXT ERROR DOERR)	Trigger given *error*.
DOLIST (PRG LIST PROC DOLIS)	Apply *program* successively to each of *n lists*, placing *results* into output list in same order. (Argument *n* may be omitted if level-1 program is—or has—just a single command or user-defined function or a name containing such.)
DOSUBS (PRG LIST PROC DOSUB)	Apply *program* to successive groups of *n* elements (from 1 to *n*, 2 to *n*+1, etc.) in given *list*; *results* go into output list in same order.
DOT (MTH VECTR DOT)	Compute dot product of two arrays of same dimension.
DRAW (◁ PLOT DRAW)	Draw plot in PICT.
DRAX (◁ PLOT DRAX)	Draw axes in PICT.
DROP (◁ DROP)	Drop *obj* from stack level 1.
DROPN (◁ STACK NXT DRPN)	Drop *n*+1 objects from stack (including argument, *n*).
DROP2 (◁ STACK NXT DROP2)	Drop two objects from the stack.
DTAG (PRG TYPE NXT DTAG)	Remove *tag* from *obj*.
DUP (◁ STACK NXT DUP)	Duplicate the level-1 *obj*. on level 2 (ENTER) works, also, if Command Line is clear).

Stack Inputs	→	Stack Outputs
1: *MM.DDYYYY*	→	
2: *MM.DDYYYY*$_A$		
1: *x*	→ 1:	*MM.DDYYYY*$_B$
2: *MM.DDYYYY*$_A$		
1: *MM.DDYYYY*$_B$	→ 1:	*days from A to B*
1: '*name*'	→ 1:	*value*$_{name}$–1
1: '*name=expr*'	→	
1: *alarm number*	→	
1: *delay*	→	
1: 'S' or { *keycode*$_1$ … … *keycode*$_n$ } or 0	→	
1: { *name*$_y$ *start*$_y$ *end*$_y$ }	→	
	→ 1:	*stack depth*
1: [[*square matrix*]]	→ 1:	*determinant*
1: *library number*	→	
2: [*diagonals*]		
1: { *n m* }	→ 1:	[[*matrix*]]
1: [[*matrix*]]	→ 1:	[*diagonals*]
2: *obj*		
1: *n*	→	
1: *t/f*	→	
1: n_{error} or "*error*"	→	
n+2: { *list*$_1$ }		
: :		
3: { *list*$_n$ }		
2: *n*		
1: « *program* »	→ 1:	{ *results* }
3: { *list* }		
2: *n*		
1: « *program* »	→ 1:	{ *results* }
2: [[*A*]]		
1: [[*B*]]	→ 1:	***A*** • ***B***
1: *obj*	→	
n+1: *obj*$_1$		
: :		
2: *obj*$_n$		
1: *n*	→	
2: *obj*$_1$		
1: *obj*$_2$	→	
1: *tag*: *obj*	→ 1:	*obj*
	2:	*obj*$_1$
1: *obj*$_1$	→ 1:	*obj*$_1$

Name (keyboard access) **Description**

DUPN (◀STACK NXT DUPN) Duplicate the objects on levels 2 through $n+1$, placing duplicates on levels $n+1$ through $2n$, respectively (after dropping original argument, n, from level 1).

DUP2 (◀STACK NXT DUPN) Duplicate the objects on levels 1 and 2, placing duplicates on levels 3 and 4, respectively.

D→R (MTH REAL NXT NXT D→R) Degrees → radians.
e (α◀E ENTER) Natural log base (symbolic constant).
EGV (MTH MATR NXT EGV) Calculate *right eigenvectors* and *eigenvalues* of *square matrix*.
EGVL (MTH MATR NXT EGV) *Eigenvalues* of *sq. mat.*
ELSE (PRG BRCH IF ELSE) Start *false* clause.
END (PRG BRCH IF END) End prgm. structure.
ENDSUB (PRG LIST PROC ENDS) # of DOSUB sublists.
ENG (◀MODES FMT ENG) Eng. mode w/$n+1$ digits.
EQ→ (PRG TYPE NXT EQ→) Separate equation into the two expressions appearing on either side of the =.
EQNLIB (◀EQ LIB EQLIB EQNLI) Start Equat. Library.
ERASE (◀PLOT ERASE) Blank out PICT (w/same dim.)
ERRM (PRG NXT ERROR ERRM) *Last error message.*
ERRN (PRG NXT ERROR ERRN) *Last error number.*
ERR0 (PRG NXT ERROR ERR0) Clear last-error data.
EVAL (EVAL) Evaluate *obj* (if it's a global name, command or program); or put contents of *obj* onto stack.
EXP (◀e^x) Exponentiate.
EXPAN (◀SYMBOLIC EXPA) Expand *expr.*
EXPFIT (◀STAT ΣPAR MODL EXPFI) Exp. curve fit.
EXPM (MTH HYP NXT EXPM) High-accuracy expon.
EYEPT (◀PLOT NXT 3D VPAR NXT EYEPT) Specify coordinates of *eyepoint* for 3-D (perspective) plot; coordinates are stored in VPAR.
F0λ (◀EQ LIB UTILS F0λ) Fraction of total blackbody emission in wavelengths 0 to λ, at temperature T.
FACT (must be typed) Factorial function (see also: !).
FANNING (◀EQ LIB UTILS F0λ) Compute *Fanning friction factor*, given *roughness* and *Reynolds number*.
FC? (◀MODES FLAG FC?) Test if given *flag* is clear.
FC?C (◀MODES FLAG FC?C) Test & clear given *flag*.
FFT (MTH NXT FFT FFT) Discrete Fourier transfrm.
FINDALARM (◀MODES FLAG FC?C) First *alarm* due after given *date* and *time*.
FINISH (◀I/O SRVR FINIS) End Kermit Serv. mode.
FIX (◀MODES FMT FIX) Display n dec. places.
FLOOR (MTH REAL NXT NXT FLOOR) Greatest int. $\leq x$.

Stack Inputs	→	Stack Outputs
	2*n*:	*obj*$_1$
	:	:
n+1:	*obj*$_1$ *n*+1:	*obj*$_n$
:	: *n*:	*obj*$_1$
2:	*obj*$_n$:	:
1:	*n* → 1:	*obj*$_n$
	4:	*obj*$_1$
	3:	*obj*$_2$
2:	*obj*$_1$ 2:	*obj*$_1$
1:	*obj*$_2$ → 1:	*obj*$_2$
1:	*θ* → 1:	*θ*(π/180)
	→ 1:	'e' or 2.7182818...
	2:	[[*rt. eigenvectors*]]
1:	[[*square matrix*]] → 1:	[*eigenvalues*]
1:	[[*square matrix*]] → 1:	[*eigenvalues*]
1:	*n* →	
	2:	'*expr*$_1$'
1:	'*expr*$_1$=*expr*$_2$' → 1:	'*expr*$_2$'
	→ 1:	"*last error message*"
	→ 1:	*last error number*
	:	(*results, if any*)
1:	*obj* → 1:	(*results, if any*)
1:	*z* → 1:	e^z
1:	'*expr*' → 1:	'*expanded expr*'
1:	*x* → 1:	e^x-1
3:	$x_{eyepoint}$	
2:	$y_{eyepoint}$	
1:	$z_{eyepoint}$ →	
2:	*λ*	
1:	*T* → 1:	*emissive power fraction*
1:	*n* → 1:	*n*!
2:	*relative roughness*	
1:	*Reynolds number* → 1:	*Fanning friction factor*
1:	*flag number* → 1:	1 or 0
1:	*flag number* → 1:	1 or 0
1:	[[*original array*]] → 1:	[[*transform array*]]
1:	{ *date time* } → 1:	*alarm index number*
1:	*n* →	
1:	*x* → 1:	FLOOR(*x*)

Name (keyboard access) **Description**

FOR (PRG BRCH FOR FOR) Begin FOR-NEXT or FOR-STEP loop program structure.

FP (MTH REAL NXT FP) Fractional portion of *x*.

FREE (must be typed) Frees the RAM of given *backup* objects or *libraries* previously merged in given port.

FREE1 (↰ LIBRARY FREE1) Like FREE but only port 1.

FREEZE (PRG NXT OUT FREEZ) Freeze *display part(s)*: 1=status area; 2=Stack/command line; 4= Menu area.

FS? (↰ MODES FLAG FS?) Test if given *flag* is set.

FS?C (↰ MODES FLAG FS?C) Test & clear given *flag*.

FUNCTION (↰ PLOT PTYPE FUNC) Plot type=Function.

GET (PRG LIST ELEM GET) Extract indicated object from given (or named) *array* or *list*.

GETI (PRG LIST ELEM GETI) Extract indicated *obj* from given (or named) *array* or *list*; return also original object and incremented position for next extraction.

GOR (PRG GROB GOR) Superimpose $grob_b$ on $grob_a$, with upper left corner of $grob_b$ at given place in $grob_a$. (If $grob_a$ is PICT, no $grob_c$ is returned.)

GRAD (↰ MODES ANGL GRAD) Set Grads angle mode.

GRAPH (must be typed) Select Picture environment.

GRIDMAP (↰ PLOT 3D PTYPE GRID) = plot type.

→GROB (PRG GROB →GRO) Create *grob* from given *obj*, using specified *character size*.

GXOR (PRG GROB GOR) Superimpose (with b/w inversion) $grob_b$ on $grob_a$, with upper left corner of $grob_b$ at given place in $grob_a$. (If $grob_a$ is PICT, no $grob_c$ returns.)

*H (↰ PLOT PPAR NXT *H) Multiply vert. plt. scale.

HALT (PRG NXT RUN HALT) Halt progr. execution.

HEAD (PRG NXT RUN HALT) Get first element.

HEX (MTH BASE HEX) Set hexadecimal base mode.

HISTOGRAM (↰ PLOT NXT STAT PTYPE HISTO) Set plot type to Histogram.

HISTPLOT (↰ STAT PLOT HISTP) Draw histogram.

HMS+ (↰ TIME NXT HMS+) Add two real numbers formatted as times; return a time-formatted result.

HMS- (↰ TIME NXT HMS-) Subtract two real numbers formatted as times; return a time-formatted result.

HMS→ (↰ TIME NXT HMS→) Formatted time → dec. hrs.

→HMS (↰ TIME NXT →HMS) Dec. hrs. → formatted time.

HOME (↱ HOME) Move to HOME directory.

i (α ↰ I) Symbolic constant, $\sqrt{(-1)}$.

IDN (MTH MATR MAKE IDN) Form *n*x*n* identity mat.

IF (PRG BRCH IF IF) Begin IF structure.

IFERR (PRG NXT ERROR IFERR) Begin IFERR trap.

IFFT (MTH NXT FFT IFFT) Inverse of FFT.

IFT (PRG BRCH NXT IFT) Evaluate *obj* if *tlf* is not zero; discard *obj* if *tlf* is zero.

Stack Inputs	→	Stack Outputs
2: *Counter begin value*		
1: *Counter end value*	→	
1: *x*	→ 1:	FP(*x*)
2: { *name*$_{backup}$ … *n*$_{library}$ }		
1: *n*$_{port}$	→	
1: { *name*$_{backup}$ … *n*$_{library}$ }	→	
1: *sum of display part(s)*	→	
1: *flag number*	→ 1:	1 or 0
1: *flag number*	→ 1:	1 or 0
2 [[*array*]] or { *list* }	→	
1: { *row col* } or *pos*	1:	*obj*
	3:	[[*array*]] or { *list* }
2: [[*array*]] or { *list* }	→ 2:	{ *row col* }$_{next}$ or *pos*$_{next}$
1: { *row col* } or *pos*	→ 1:	*obj*
3: *grob*$_a$		
2: { #*n* #*m* } or (*x*, *y*)		
1: *grob*$_b$	→ 1:	*grob*$_c$
2: *obj*		
1: *character size*	→ 1:	*grob*
3: *grob*$_a$		
2: { #*n* #*m* } or (*x*, *y*)		
1: *grob*$_b$	→ 1:	*grob*$_c$
1: *vertical scale factor*	→	
1: { *list* } or "*string*"	→ 1:	*element*$_i$ or "*char*$_i$"
2: *HH.MMSSs…*$_A$		
1: *HH.MMSSs…*$_B$	→ 1:	*HH.MMSSs…*$_{A+B}$
2: *HH.MMSSs…*$_A$		
1: *HH.MMSSs…*$_B$	→ 1:	*HH.MMSSs…*$_{A-B}$
1: *HH.MMSSs…*	→ 1:	*HH.hhhhh…*
1: *HH.hhhhh…*	→ 1:	*HH.MMSSs…*
	→ 1:	'i' or (0, 1)
1: *n* or [[*sq. matrix*]]	→ 1:	[[*identity matrix*]]
1: [[*transform array*]]	→ 1:	[[*original array*]]
2: *tlf*	:	(varies with *tlf* and *obj*)
1: *obj*	→ 1:	(varies with *tlf* and *obj*)

Name (keyboard access)	Description
IFTE (PRG BRCH NXT IFTE)	Evaluate obj_t if *tlf* is not zero; evaluate obj_f if *tlf* is zero.
IM (MTH NXT CMPL IM)	Imaginary part of complx.
INCR (MEMORY ARITH INCR)	Increment the *named* value by 1; echo the *new value* to stack.
INDEP (PLOT PPAR INDEP)	Specify the independent variable *name* and (optionally) the plotting range.
INFORM (PRG NXT IN INFOR)	Create custom input form; field definition def_n is { "*label*" "*prompt*" $type_1$ $type_2$ ... } ({ } is OK); *cols* is the # of columns, *tabs* is the spacing from label to field ({ } is OK); $reset_n$ and $initial_n$ are *field*$_n$'s reset and initial values (NOVAL is OK).
INPUT (PRG NXT IN INPUT)	Prompt for input on command line; lock out stack operations.
INV (1/x)	Compute reciprocal (or matrix inverse).
IP (MTH REAL NXT IP)	Integer portion of x.
IR (I/O IOPAR IR)	Toggle I/O (infrared/serial).
ISOL (SYMBOLIC ISOL)	Rearrange *equation* to isolate first occurrence of *name*.
KERRM (I/O NXT KERR)	Most recent Kermit error.
KEY (PRG NXT IN KEY)	Return 1 and the *keycode* location if a key is pressed; return 0 if no key is pressed.
KGET (I/O SRVR KGET)	Get object via Kermit.
KILL (PRG NXT RUN KILL)	Cancel halted program.
LABEL (PLOT NXT LABEL)	Label axes in PICT.
LASTARG or LAST (ARG)	Return arguments of most recently executed command.
LCD→ (PRG GROB NXT LCD→)	Make *grob* from display.
→LCD (PRG GROB NXT LCD→)	Display given *grob*.
LIBEVAL (must be typed)	Do unlisted library command.
LIBS (LIBRARY LIBS)	Libraries attached to curr. dir.
LINE (PRG PICT LINE)	Draw line in PICT between given coordinates
ΣLINE (STAT FIT ΣLINE)	Best fit w/curr. model.
LINFIT (STAT ΣPAR MODL LINFI)	Linear model.
LININ (PRG TEST PREV LININ)	Test if given *expr* is linear in *named* variable.
LIST→ (must be typed)	Decompose given list into its constituent objects, placing these on the stack in order, followed by index, n, indicating size of original list (see also OBJ→).
→LIST (PRG TYPE →LIST)	Construct a list using, in order, the first n objects found on the stack (after first removing argument n).

Stack Inputs	→	Stack Outputs
3: *tlf*		
2: obj_1		
1: obj_2	→ 1:	(varies)
1: (x, y)	→ 1:	y
1: '$name_{obj}$'	→ 1:	*new value*$_{obj}$
2: *start* (optional)		
1: { *name start end* }	→	
5: "*title*"		
4: { def_1 def_2 ... }		
3: { *cols tabs* }		
2: { $reset_1$ $reset_2$ ... }	2:	{ $value_1$ $value_2$ ... }
1: { $initial_1$ $initial_2$ ... }	→ 1:	1 or 0
2: "*prompt*"		
1: "*default value*"	→ 1:	"*result*"
1: z	→ 1:	1/z
1: x	→ 1:	n
2: '*equation*'		
1: '*name*'	→ 1:	'*expr*'
	→ 1:	"*error message*"
	2:	*keycode*$_{row\ column}$
	→ 1:	1 or 0
1: '*name*'	→	
	n:	obj_n
	:	
	2:	obj_2
	→ 1:	obj_1
	→ 1:	*grob* (131 x 64)
1: *grob*	→	
1: #$n_{ccellll}$	→ 1:	(depends)
	→ 1:	{ "$title_1$" lib_1 $port_1$... }
2: { #n_1 #m_1 }		
1: { #n_2 #m_2 }	→	
	→ 1:	'$expr_{best\ fit}$'
2: '*expr*'		
1: '*name*'	→ 1:	1 or 0
	n+1:	obj_1
	:	
	2:	obj_n
1: { obj_1 obj_2... obj_n }	→ 1:	n
n+1: obj_1		
:		
2: obj_n		
1: n	→ 1:	{ obj_1 obj_2... obj_n }

Name (keyboard access)	Description
ΣLIST ((MTH) LIST ΣLIST)	List elements' sum.
ΔLIST ((MTH) LIST ΔLIST)	List elements' differences.
ΠLIST ((MTH) LIST ΠLIST)	List elements' product.
LN ((→) LN)	Natural (base *e*) logarithm.
LNP1 ((MTH) HYP (NXT) LNP1)	Better LN for $x \approx 0$.
LOG ((→) LOG)	Common (base 10) logarithm.
LOGFIT ((←) STAT ΣPAR MODL LOGFI)	LR model=log.
LQ ((MTH) MATR FACTR LQ)	Factor the given matrix, A, into 3 other matrices, L, Q, and P, such that $P \times A = L \times Q$; L is lower-trapezoidal, and Q is orthogonal.
LR ((←) STAT FIT LR)	Compute linear regression coefficients using currently selected fit model.
LSQ ((MTH) MATR LSQ)	Find minimum-norm least-squares solution to linear system $A \times X = B$.
LU ((MTH) MATR FACTR LU)	Factor the given matrix, A, into 3 other matrices, L, U, and P, such that $P \times A = L \times U$; L is lower-triangular, and U is upper-triangular.
MANT ((MTH) REAL (NXT) MANT)	Mantissa of argument.
↑MATCH ((←) SYMBOLIC (NXT) ↑MAT)	Replace occurrences of *pat* with *repl* in $expr_1$, starting with innermost clause.
↓MATCH ((←) SYMBOLIC (NXT) ↓MAT)	Replace occurrences of *pat* with *repl* in $expr_1$, starting with outermost clause.
MAX ((MTH) REAL MAX)	Find the greater of two given arguments.
MAXR ((MTH) (NXT) CONS (NXT) MAXR)	Greatest absolute value that can be represented in machine.
MAXΣ ((←) STAT ΣVAR MAXΣ)	ΣDAT column *maxima*.
MCALC ((←) EQ LIB MES MCAL)	MES calc. values.
MEAN ((←) STAT ΣVAR MEAN)	ΣDAT column *means*.
MEM ((←) MEMORY MEM)	Bytes of *RAM* now available.
MENU ((←) MODES MENU MENU)	Display indicated menu.
MERGE (must be typed)	Merge port 1 RAM & main RAM.
MERGE1 ((←) LIBRARY MERG)	Like MERGE, but no arg.
MIN ((MTH) REAL MIN)	Find the lesser of two given arguments.
MINEHUNT ((←) EQ LIB UTILS MINE)	Minehunt game.
MINIT ((←) EQ LIB MES MINIT)	Create Mpar for MES.
MINR ((MTH) (NXT) CONS (NXT) MINR)	Least non-zero absolute value that can be represented in machine.
MINΣ ((←) STAT ΣVAR MINΣ)	ΣDAT column *minima*.
MITM ((←) EQ LIB MES MITM)	Change title and menu order of Multiple Equation Solver (MES).
MOD ((MTH) REAL MOD)	Compute modulo remainder of x/y, defined as $x \bmod y = x - y \cdot \text{floor}(x/y)$.
MROOT ((←) EQ LIB MES MROO)	Find a root via MES.
MSGBOX ((PRG) (NXT) OUT MSGB)	Show message box.
MSOLVR ((←) EQ LIB MES MSOL)	Get MES menu.
MUSER ((←) EQ LIB MES MUSE)	MES input values.

Stack Inputs	→	Stack Outputs
1: { ele_1 ele_2 ele_3 ... }	→ 1:	ele_1+ele_2+ele_3+ ...
1: { ele_1 ele_2 ele_3 ... }	→ 1:	{ ele_2–ele_1 ele_3–ele_2,... }
1: { ele_1 ele_2 ele_3 ... }	→ 1:	ele_1•ele_2•ele_3• ...
1: z	→ 1:	ln(z)
1: x	→ 1:	ln(x+1)
1: z	→ 1:	log(z)
	3:	[[***L*** (m x n)]]
	2:	[[***Q*** (n x n)]]
1: [[***A*** (m x n)]]	→ 1:	[[***P*** (m x m)]]
	2:	*intercept*
	→ 1:	*slope*
2: [[***B***]] or [***B***]		
1: [[***A***]]	→ 1:	[[***X***]] or [***X***]
	3:	[[***L***]]
	2:	[[***U***]]
1: [[***A*** (n x n)]]	→ 1:	[[***P***]]
1: x	→ 1:	*mantissa*
2: '$expr_1$'	2:	'$expr_2$'
1: { '*pat*' '*repl*' }	→ 1:	1 or 0
2: '$expr_1$'	2:	'$expr_2$'
1: { '*pat*' '*repl*' }	→ 1:	1 or 0
2: a		
1: b	→ 1:	max(a,b)
		'MAXR' or
	→ 1:	9.99999999999E499
	→ 1:	{ max_{col1} max_{col2} ... }
1: { $name_1$ $name_2$ ... }	→	
	→ 1:	{ $mean_{col1}$ $mean_{col2}$... }
	→ 1:	*RAM*
1: *menu number*	→	
1: 1	→	
2: a		
1: b	→ 1:	min(a,b)
	→ 1:	'MINR' or 1E-499
	→ 1:	{ min_{col1} min_{col2} ... }
2: "*title*"		
1: { $name_1$ $name_2$ ... }	→	
2: x		
1: y	→ 1:	x mod y
1: '*name*'	→ 1:	x
1: "*message*"	→	
1: { $name_1$ $name_2$ ... }	→	

Name (keyboard access) **Description**

NDIST (MTH NXT PROB NXT NDIST) Compute normal probability distribution at data value x, assuming mean m and variance v.

NEG (+/–) Negate or change sign of argument.

NEWOB (MEMORY NEWOB) Create new version of given *obj* (so that original version may be purged).

NEXT (PRG BRCH FOR NEXT) End FOR or START loop structure.

NOT (PRG TEST NXT NOT) Logical opposite of arg.

NOVAL (PRG NXT IN NOVAL) Empty field in INFORM.

NSUB (PRG LIST PROC NSUB) DOSUBS sublist pos.

NUM (CHARS NUM) Code of first character in *string*.

→NUM (→NUM) Numeric value of symbolic *arg*.

NUMX (PLOT NXT 3D VPAR NXT NUMX) Set the number of *x-steps* per *y*-step in 3D perspective plots.

NUMY (PLOT NXT 3D VPAR NXT NUMY) Set the no. of *y-steps* in view volume of 3D perspective plot.

NΣ (STAT SUMS NΣ) Number of *rows* in ΣDAT.

OBJ→ (PRG TYPE OBJ→) Separate an object into its components and place those components, in order, on the stack. (For some object types, such as a matrix, vector or list—shown here—the dimension or number of components is returned to level 1.)

OCT (MTH BASE OCT) Set octal displ. for binary int.

OFF (PRG NXT RUN NXT OFF) Turn off calculator.

OLDPRT (I/O PRINT PRTPA OLDPR) Adjust PRTPAR to prepare for IR printer.

OPENIO (I/O NXT SERIA OPENI) Open serial/IR port.

OR (PRG TEST NXT OR) Logical OR of two given arguments.

ORDER (MEMORY DIR ORDER) Reorder VAR menu.

OVER (STACK OVER) Make copy of level-2 object and return the copy to level 1.

PARAMETRIC (PLOT PTYPE PARA) Parametric ptype

PARITY (I/O IOPAR PARIT) Set *parity* in IOPAR.

PARSURFACE (PLOT NXT 3D PTYPE PARSU) Set plot type to Parametric Surface.

PATH (MEMORY DIR PATH) Path of current dir.

PCOEF (SOLVE POLY PCOEF) Compute monomic polynomial with given real or complex *roots*.

PCONTOUR (PLOT NXT 3D PTYPE PCON) Set the plot type to PCONTOUR.

PCOV (STAT FIT NXT PCOV) ΣDAT *pop. covar.*

PDIM (PRG PICT PDIM) Create a blank grob of given dimensions and store it in PICT.

PERM (MTH NXT PROB PERM) Compute the number of possible permutations of n items taken k at a time.

Stack Inputs	→	Stack Outputs
3: *m*		
2: *v*		
1: *x*	→ 1:	$\exp[-(x-m)^2/2v]/\sqrt{2\pi v}$
1: *z*	→ 1:	*-z*
1: *obj*	→ 1:	*obj*
1: *t/f*	→ 1:	0 or 1
1: "*string*"	→ 1:	*character code*
1: '*arg*'	→ 1:	*z*
1: *x-steps*	→	
1: *y-steps*	→	
	→ 1:	*rows*
	n+1:	obj_1
	n:	obj_2
	:	
	2:	obj_n
1: { obj_1 obj_2 ... obj_n }	→ 1:	*n*
2: t/f_A		
1: t/f_B	→ 1:	1 or 0
1: { $name_1$ $name_2$ ... }	→	
	3:	obj_1
2: obj_1	2:	obj_2
1: obj_2	→ 1:	obj_1
1: *parity*	→	
	→ 1:	{ HOME dir_1 dir_2 ... }
1: [$root_1$ $root_2$ ...]	→ 1:	[1 $coeff_{n-1}$ $coeff_{n-2}$ ...]
	→ 1:	*population covariance*
2: (x_{min}, y_{min})		
1: (x_{max}, y_{max})	→	
2: *n*		
1: *k*	→ 1:	$P_{n,k}$

Name (keyboard access) **Description**

PEVAL (◁SOLVE **POLY** **PEVAL**) Evaluate polynomial with given *coeff*icients at given value, *x*.
PGDIR (◁MEMORY **DIR** **PGDIR**) Purge directory.
PICK (◁STACK **PICK**) Make a copy of the object found at the stack level indicated (after removing argument *n*); put this copy onto the stack.
PICT (PRG **PICT** **PICT**) Put PICT on stack.
PICTURE (◁PICTURE) Select graphics display & menu.
PINIT (◁LIBRARY NXT **PINIT**) Initialize all active ports.
PIXOFF (PRG **PICT** NXT **PIXOF**) Turn off pixel in PICT.
PIXON (PRG **PICT** NXT **PIXON**) Turn on pixel in PICT.
PIX? (PRG **PICT** NXT **PIX?**) Test pixel in PICT.
PKT (◁I/O **SRVR** **PKT**) Send command packets to, and receive requested *data* from, a Kermit server.
PMAX (must be typed) Set upper right limits of PICT.
PMIN (must be typed) Set lower left limits of PICT.
POLAR (◁PLOT **PTYPE** **POLAR**) Set plot type to Polar.
POS (◁CHARS **POS**) Find *position* of *obj* in *list* or *substring* in *string*.
PREDV (must be typed) Same as PREDY.
PREDX (◁STAT **FIT** **PREDX**) Find indep. value, *x*, given dep. value, *y*, and current ΣDAT and LR model.
PREDY (◁STAT **FIT** **PREDY**) Find depend. value, *y*, given indep. value, *x*, and current ΣDAT and LR model.
PRLCD (◁I/O **PRINT** **PRLCD**) Print image of display.
PROMPT (PRG NXT **IN** NXT **PROM**) Show *prompt* in status area and halt program execution.
PROOT (◁SOLVE **POLY** **PROOT**) Roots of polynomial.
PRST (◁I/O **PRINT** **PRST**) Print all stack objects.
PRSTC (◁I/O **PRINT** **PRSTC**) Compact format of PRST.
PRVAR (◁I/O **PRINT** **PRVAR**) Print given variables.
PR1 (◁I/O **PR1**) Print *obj* in multi-line format.
PSDEV (◁STAT **1VAR** NXT **PSDEV**) ΣDAT *pop. sdevs.*
PURGE (◁PURG) Purge given vars. and (empty) dirs.
PUT (PRG **LIST** **ELEM** **PUT**) Replace *position* in given *list* (or given { *row col* } in given array) with given *obj*; return result if original was unnamed.
PUTI (PRG **LIST** **ELEM** **PUTI**) Replace *position* in given *list* (or given { *row col* } in given array) with given *obj*; return result (or original *name*) and next *position*.
PVAR (◁STAT **1VAR** NXT **PVAR**) ΣDAT *pop. varnces.*
PVARS (◁LIBRARY **PVARS**) List of variables in specified port, *p*, and its available RAM *memory*, if applicable.
PVIEW (PRG **PICT** NXT **PVIEW**) Display PICT with its specified coordinate at upper left of display.
PWRFIT (◁STAT **SPAR** **MODL** **PWRFI**) LR mdl.=Power.

Stack Inputs	→	Stack Outputs
2: [$coeff_n$ $coeff_{n-1}$ …]		
1: x	→ 1:	$p(x)$
1: '*name*'	→	
$n+1$: obj_n	$n+1$:	obj_n
n: obj_{n-1}	n:	obj_{n-1}
:	:	
2: obj_1	2:	obj_1
1: n	→ 1:	obj_0
	→ 1:	PICT
1: { #*n* #*m* } or (x, y)	→	
1: { #*n* #*m* } or (x, y)	→	
1: { #*n* #*m* } or (x, y)	→ 1:	0 or 1
2: "*data*"		
1: "*type*"	→ 1:	"*response*"
1: (x, y)		
1: (x, y)		
2: { *list* } or "*string*"		
1: *obj* or "*substring*"	→ 1:	*position*
1: x	→ 1:	y
1: y	→ 1:	x
1: x	→ 1:	y
1: "*prompt*"	→	
1: [$coeff_n$ $coeff_{n-1}$ …]	→ 1:	[$root_1$ $root_2$ …]
1: { $name_1$ $name_2$ … }	→	
1: *obj*	→	
	→ 1:	[psd_1 psd_2 … psd_m]
1: { $name_1$ $name_2$ … }	→	
3: { $list_A$ } or '*name*'		
2: *position*		
1: *obj*	→ 1:	{ $list_B$ } (or nothing)
3: { $list_A$ } or '*name*'		
2: *position*	2:	{ $list_A$ } or '*name*'
1: *obj*	→ 1:	*position*+1
	→ 1:	[$pvar_1$ $pvar_2$ … $pvar_m$]
	2:	{ :p:n_{lib1} … :p:n_{libk}]
1: p	→ 1:	*memory*
1: { #*n* #*m* } or (x, y)	→	

Name (keyboard access)	Description
PX→C (PRG PICT NXT PX→C)	Pixel coord.→user units.
→Q (↶SYMBOLIC NXT →Q)	Rationalize argument.
→Qπ (↶SYMBOLIC NXT →Qπ)	Rationalize with π factor.
QR (MTH MATR FACTR QR)	Factor the given matrix, A, into 3 other matrices, Q, R, and P, such that $A \times P = Q \times R$; R is upper-trapezoidal, and Q is orthogonal.
QUAD (↶SYMBOLIC →Qπ)	Solve $expr_A$ for *name*, using 2nd-deg. Taylor expans. to convert $expr_A$ to quadratic.
QUOTE (↶SYMBOLIC NXT NXT QUOT)	Unevaluated *arg*.
RAD (↶RAD)	Set Radians mode.
RAND (MTH NXT PROB RAND)	Get pseudo-random #.
RANK (MTH MATR NORM NXT RANK)	Rank of *matrix*.
RANM (MTH MATR MAKE RANK)	Create random mat.
RATIO (must be typed)	Algebraic form of ⁄ (division) function.
RCEQ (↶PLOT EQ)	Recall contents of EQ.
RCI (MTH MATR ROW RCI)	Multiply indicated row n of $array_A$ by *factor* (note that vectors are treated as column matrices).
RCIJ (MTH MATR ROW RCIJ)	Multiply row i of given $array_A$ by *factor*, then add result to row j (note that vectors are treated as column matrices).
RCL (↷RCL)	Recall *named* object to stack.
RCLALARM (↶TIME ALRM RCLAL)	Recall given alarm; result has *action* taken & *int*erval in ticks (1/8192 sec).
RCLF (↶MODES FLAG NXT RCLF)	Recall *flags*.
RCLKEYS (↶MODES KEYS RCLK)	Rcl. assigned *keys*.
RCLMENU (↶MODES MENU RCLM)	Rcl. current *menu*.
RCLΣ (↶PLOT NXT STAT ΣDAT)	Recall ΣDAT.
RCWS (MTH BASE NXT RCWS)	Rcl. curr. *bin. wordsize*.
RDM (MTH MATR MAKE RDM)	Redimension $array_A$ as specified (if $array_A$ is named, nothing is returned).
RDZ (MTH NXT PROB RDZ)	New *seed* for RAND.
RE (MTH NXT CMPL RE)	Real part of argument
RECN (↶I/O NXT RECN)	Receive & *name* Kermit file.
RECT (↶MODES ANGL RECT)	Set Rectangular mode.
RECV (↶I/O RECV)	Receive Kermit file already named.
REPEAT (PRG BRCH WHILE REPEA)	WHILE loop clause.
REPL (PRG LIST REPL)	Replace part of obj_A with *repl* object, beginning at specified *position*. The obj_A and *repl* objects may be strings, arrays, lists, or grobs.
RES (↶PLOT PPAR RES)	Set plot resolution.
RESTORE (↶MEMORY NXT RESTO)	Get HOME dir backup.
REVLIST (MTH LIST REVLI)	Reverse list elements.
RKF (↶SOLVE DIFFE RKF)	Find solution to initial-value differential *equation*, via Runge-Kutta-Fehlberg (4,5), given *tolerance*, *final* value & *names* of variables.

Stack Inputs	→	Stack Outputs
1: { #*n* #*m* }	→ 1:	(x_9 *y*)
1: *x*	→ 1:	'*a*↙*b*'
1: *x*	→ 1:	'*a*↙*b**π'
	3:	[[***Q*** (m x m)]]
	2:	[[***R*** (m x n)]]
1: [[***A*** (m x n)]]	→ 1:	[[***P*** (n x n)]]
2: '*expr*$_A$'		
1: '*name*'	→ 1:	'*expr*$_B$'
1: *arg*	→ 1:	*arg*
	→ 1:	$0 \le x_{random} \le 1$
1: [[*matrix*]]	→ 1:	*rank*
1: { *m n* }	→ 1:	[[*matrix*$_{random\ integers}$]]
2: z_1		
1: z_2	→ 1:	z_1/z_2
	→ 1:	*obj*$_{EQ}$
3: [*array*$_A$]		
2: *factor*		
1: *n*	→ 1:	[*array*$_B$]
4: [*array*$_A$]		
3: *factor*		
2: *i*		
1: *j*	→ 1:	[*array*$_B$]
1: '*name*'	→ 1:	*obj*$_{name}$
1: *alarm number*	→ 1:	{ *date time obj*$_{action}$ *int* }
	→ 1:	{ #*flags*$_{system}$ #*flags*$_{user}$ }
	→ 1:	{ *obj*$_1$ *key*$_1$ *obj*$_2$ *key*$_2$... }
	→ 1:	*menu number*
	→ 1:	*obj*$_{\Sigma DAT}$
	→ 1:	*wordsize* (bits)
2: [*array*$_A$]		
1: { *n* } or { *m n* }	→ 1:	[*array*$_B$]
1: *random seed value*	→	
1: (x_9 *y*)	→ 1:	*x*
1: '*name*'	→	
1: *t/f*	→	
3: *obj*$_A$		
2: *position*		
1: *repl*	→ 1:	*obj*$_B$
1: *x-interval*	→	
1: :*port*:*name*$_{backup}$	→	
1: { *obj*$_1$ *obj*$_2$ ... *obj*$_n$ }	→ 1:	{ *obj*$_n$ ... *obj*$_2$ *obj*$_1$ }
3: { *name*$_t$ *name*$_y$ *eq*$_{right}$ }		
2: *abs. error tolerance*	2:	{ *name*$_t$ *name*$_y$ *eq*$_{right}$ }
1: t_{final}	→ 1:	*abs. error tolerance*

Name (keyboard access) **Description**

RKFERR ((←)SOLVE DIFFE RKFE) Estimate absolute *error* for given *step* when solving initial-value differential *eq*uation, via Runge-Kutta-Fehlberg, given (right-hand side of) *eq*uation and *names* of variables.

RKFSTEP ((←)SOLVE DIFFE RKFS) Find next Runge-Kutta-Fehlberg solution *step* of initial-value differential *eq*uation, given variable *names*, $step_{initial}$, and *tolerance*.

RL ((MTH) BASE (NXT) BIT RL) Rotate left 1 bit.

RLB ((MTH) BASE (NXT) BYTE RLB) Rotate left 1 byte.

RND ((MTH) REAL (NXT)(NXT) RND) Round $value_A$ to n decimal places (for $0 \le n \le 11$); or to n significant digits (for $n < 0$); or to current display setting (for $n = 12$).

RNRM ((MTH) MATR NORM RNRM) Row norm of *array*.

ROLL ((←)STACK ROLL) Rotate the contents of the bottom n levels of the stack (after consuming argument n) by rolling each *obj* up a level (except for obj_n, which goes to level 1).

ROLLD ((←)STACK ROLLD) Rotate the contents of the bottom n levels of the stack (after consuming argument n) by rolling each *obj* down a level (except for obj_1, which goes to level n).

ROOT ((←)SOLVE ROOT ROOT) Find *root* value of given variable—i.e. that which gives result nearest zero in the given *prgm* or *expr*, starting with the given *guess(es)*.

ROT ((←)STACK ROT) Rotate the contents of the bottom 3 levels of the stack by rolling each *obj* up a level (except for obj_3, which goes to level 1).

→ROW ((MTH) MATR ROW →ROW) Decompose given *array* into *rows*, placing *rows* in order on the stack, followed by the *row count*. Vectors are treated as column matrices.

ROW+ ((MTH) MATR ROW ROW+) Insert $array_B$ into $array_A$, beginning at indicated *position*.

ROW- ((MTH) MATR ROW ROW-) Delete row n from given $array_A$; return resulting $array_B$ and deleted row_n.

ROW→ ((MTH) MATR ROW →ROW) Build *array* from n given *rows*. Vectors are treated as column matrices.

RR ((MTH) BASE (NXT) BIT RR) Rotate right 1 bit.

RRB ((MTH) BASE (NXT) BYTE RRB) Rotate rt. 1 byte.

RREF ((MTH) MATR FACTR RREF) Convert a (square) *matrix* to its reduced row echelon form.

Stack Inputs	→	Stack Outputs
	4:	{ $name_t$ $name_y$ eq_{right} }
	3:	*step*
2: { $name_t$ $name_y$ eq_{right} }	2:	Δy_{step}
1: *step*	→ 1:	$error_{step}$
3: { $name_t$ $name_y$ eq_{right} }	3:	{ $name_t$ $name_y$ eq_{right} }
2: *abs. error tolerance*	2:	*abs. error tolerance*
1: $step_{initial}$	→ 1:	$step_{next}$
1: $\#n_A$	→ 1:	$\#n_B$
1: $\#n_A$	→ 1:	$\#n_B$
2: $value_A$		
1: *n*	→ 1:	$value_B$
1: [*array*]	→ 1:	*row norm*
n+1: obj_n		
n: obj_{n-1}	*n*:	obj_{n-1}
:	:	
2: obj_1	2:	obj_1
1: *n*	→ 1:	obj_n
n+1: obj_n		
n: obj_{n-1}	*n*:	obj_1
:	:	
2: obj_1	2:	obj_3
1: *n*	→ 1:	obj_2
3: 'expr' or « prgm »		
2: 'name'		
1: { *guess(es)* }	→ 1:	*root*
3: obj_3	3:	obj_2
2: obj_2	2:	obj_1
1: obj_1	→ 1:	obj_3
	n+1:	[row_1]
	n:	[row_2]
	:	
	2:	[row_n]
1: [*array*]	→ 1:	*row count*
3: [$array_A$]		
2: [$array_B$]		
1: *position*	→ 1:	[$array_C$]
2: [$array_A$]	2:	[$array_B$]
1: *n*	→ 1:	[row_n]
n+1: [row_1]		
n: [row_2]		
:		
2: [row_n]		
1: *n*	→ 1:	[*array*]
1: $\#n_A$	→ 1:	$\#n_B$
1: $\#n_A$	→ 1:	$\#n_B$
1: [[*matrix*]]	→ 1:	[[$matrix_{RRE}$]]

Name (keyboard access)	Description

RRK (◀SOLVE DIFFE RRK) Find solution to initial-value differential *equation* via Rosenbrock, Runge-Kutta, given *tolerance*, *final* value, the variable *names* and their *partial derivatives*.

RRKSTEP (◀SOLVE DIFFE RRKS) Find next solution *step* of an initial-value differential *equation*, using either *method* (RKF =1; RRK =2), given the variable *names* and their *partial derivatives*, and the $step_{initial}$, and *tolerance*.

RSBERR (◀SOLVE DIFFE RSBER) Estimate absolute *error* for given $step_{initial}$ when solving initial-value differential *equation*, via Rosenbrock method, given (right side of) *equation* and the *names* and *partial derivatives* of the variables.

RSD (MTH MATR NXT RSD) Compute residual of arrays $\boldsymbol{B}$, $\boldsymbol{A}$ and $\boldsymbol{Z}$, defined as $\boldsymbol{B} - \boldsymbol{AZ}$. (If $\boldsymbol{B}$ and $\boldsymbol{Z}$ are vectors, the solution will be a vector, also.)

RSWP (MTH MATR ROW NXT RSWP) Swap indicated rows i and j of given $array_A$ (note that vectors are treated as column matrices).

R→B (MTH BASE R→B) Convert real integer to binary.

R→C (MTH NXT CMPL R→C) Convert two real numbers (or arrays) to one complex number (or array).

R→D (MTH REAL NXT NXT R→D) Convert rad. to deg.

SAME (PRG TEST NXT SAME) Test if two *objects* are identical (same as == except with symbolic objects).

SBRK (◀I/O NXT SERIA SBRK) Interrupt serial trans.

SCALE (◀PLOT PPAR NXT SCALE) Set horizontal and vertical plot *scales* (adjusts first 2 PPAR parameters).

SCATRPLOT (◀STAT PLOT SCATR) Draw scatterplot.

SCATTER (◀PLOT NXT STAT PTYPE SCATT) Set Ptype.

SCHUR (MTH MATR FACTR SCHUR) Find Schur decomposition of *matrix* $\boldsymbol{A}$, so that $\boldsymbol{A} = \boldsymbol{Q} \times \boldsymbol{T} \times \mathrm{TRN}(\boldsymbol{Q})$.

SCI (◀MODES FMT SCI) Scientif. disp. sig. digits.

SCLΣ (must be typed) Autoscale PPAR for scatterplot.

SCONJ (◀MEMORY ARITH NXT SCONJ) Conjugate value.

SDEV (◀STAT 1VAR SDEV) ΣDAT (sample) *st. devs.*

SEND (◀I/O SEND) Copy objects to Kermit device.

SEQ (PRG LIST PROC NXT SEQ) Repeatedly evaluate *obj*, using a (globally or locally) named *index* value that ranges from *start* to *finish*, varying by the indicated *increment* for each repetition; return *results* (in order) in a list.

SERVER (◀▶) Select Kermit Server mode.

SF (◀MODES FLAG SF) Set indicated *flag*.

SHOW (◀SYMBOLIC SHOW) Explicitly show all implicit occurrences of *name* in *expr*.

SIDENS (◀EQ LIB UTILS SIDEN) Silicon intr. *density*.

Stack Inputs	→	Stack Outputs
3: { $name_t$ $name_y$ eq_{right} $part.der_t$ $part.der_y$ }		2: { $name_t$ $name_y$ eq_{right} $part.der_t$ $part.der_y$ }
2: $tolerance$		
1: t_{final}	→	1: $tolerance$
4: { $name_t$ $name_y$ eq_{right} $part.der_t$ $part.der_y$ }		4: { $name_t$ $name_y$ eq_{right} $part.der_t$ $part.der_y$ }
3: $tolerance$		3: $tolerance$
2: $step_{initial}$		2: $step_{next}$
1: $method_{last}$	→	1: $method_{current}$
		4: { $name_t$ $name_y$ eq_{right} $part.der_t$ $part.der_y$ }
2: { $name_t$ $name_y$ eq_{right} $part.der_t$ $part.der_y$ }		3: $step_{initial}$
		2: Δy_{step}
1: $step_{initial}$	→	1: $error$
3: [[B]]		
2: [[A]]		
1: [[Z]]	→	1: [[$B - AZ$]]
3: [$array_A$]		
2: i		
1: j	→	1: [$array_B$]
1: n	→	1: #n
2: x		
1: y	→	1: (x, y)
1: θ	→	1: (180/π)θ
2: obj_A		
1: obj_B	→	1: 1 or 0
2: $scale_x$		
1: $scale_y$	→	
		2: [[Q]]
1: [[A (square)]]	→	1: [[T]]
1: n	→	
1: 'name'	→	
	→	1: [$stdev_1$ $stdev_2$... $stdev_m$]
1: [$name_1$ $name_2$...]	→	
5: obj		
4: 'index'		
3: $start$		
2: $finish$		
1: $increment$	→	1: { $result_1$ $result_2$... }
1: $flag\ number$	→	
2: 'expr'		
1: 'name'	→	1: 'expr$_{expanded}$'
1: $temperature$ (K)	→	1: $density_{silicon}$ (cm^{-3})

Name (keyboard access)	Description
SIGN (MTH REAL NXT SIGN)	Sign (±) of a number.
SIMU (↶ PLOT NXT FLAG SIMU)	Toggle Simultaneous Plotting mode on/off.
SIN (SIN)	Sine of given argument.
SINH (MTH REAL SINH)	Hyperbolic sine of arg.
SINV (↶ MEMORY ARITH NXT SINV)	Replace *named* value with its inverse.
SIZE (↶ CHARS SIZE)	Return the dimensions, length, or number of elements in the given *obj*.
SL (MTH BASE NXT BIT SL)	Shift left 1 bit.
SLB (MTH BASE NXT BYTE SLB)	Shift left 1 byte.
SLOPEFIELD (↶ PLOT NXT 3D PTYPE SLOPE)	Set plot type to Slopefield.
SNEG (↶ MEMORY ARITH NXT SNEG)	Replace *named* value with its negative.
SNRM (MTH MATR NORM SNRM)	*Array spectral norm.*
SOLVEQN (↶ EQ LIB EQLIB SOLVE)	Start the solver for equation with the given *subject no.* and *title no.*; put that equation's picture into PICT if indicated by level-1 value.
SORT (MTH LIST SORT)	Sort *list* elem. (ascending).
SPHERE (↶ MODES ANGL SPHER)	Set spherical mode.
SQ (↶ x²)	Square the given argument.
SR (MTH BASE NXT BIT SR)	Shift right 1 bit.
SRAD (MTH MATR NORM SNRM)	*Matrix spect. radius.*
SRB (MTH BASE NXT BYTE SRB)	Shift right 1 byte.
SRECV (↶ I/O NXT SERIA SRECV)	Read *string* of up to *n* characters from serial buffer; indicate if error occurs.
START (PRG BRCH START START)	Begin definite loop structure.
STD (↶ MODES FMT STD)	Standard display mode.
STEP (PRG BRCH START STEP)	End definite loop structure and define *counter increment*.
STEQ (↶ PLOT ↶ EQ)	Store *obj* into EQ.
STIME (↶ I/O NXT SERIA STIME)	SRECV/XMIT timeout.
STO (STO)	Store *obj* into given *name*.
STOALARM (↶ TIME ALARM STOAL)	Store given alarm.
STOF (↶ MODES FLAG NXT STOF)	Set *flags* as indic.
STOKEYS (↶ MODES KEYS STOK)	Assign *keys* as ind.
STO+ (↶ MEMORY ARITH STO+)	Add *obj* to *named* value (or vice versa); store result into *name*.
STO- (↶ MEMORY ARITH STO-)	Subtract *obj* from the *named* value (or vice versa); store result into *name*.
STO* (↶ MEMORY ARITH STO*)	Multiply *obj* by *named* value (or vice versa); store result into *name*.
STO/ (↶ MEMORY ARITH STO/)	Divide *obj* into *named* value (or vice versa); store result into *name*.
STOΣ (↶ STAT DATA ↶ ΣDAT)	Store *obj* into ΣDAT.

Stack Inputs	→	Stack Outputs
1: *x*	→ 1:	1 or 0 or -1
1: *z*	→ 1:	sin(*z*)
1: *z*	→ 1:	sinh(*z*)
1: '*name*'	→	
	2:	#*n* (or nothing)
1: *obj*	→ 1:	*n* or { *n m* } or #*m*
1: #n_A	→ 1:	#n_B
1: #n_A	→ 1:	#n_B
1: '*name*'	→	
1: [*array*]	→ 1:	*spectral norm*
3: *subject number*		
2: *title number*		
1: 1 or 0	→	
1: { $list_A$ }	→ 1:	{ $list_B$ }
1: *z*	→ 1:	z^2
1: #n_A	→ 1:	#n_B
1: [[*matrix* (square)]]	→ 1:	*spectral radius*
1: #n_A	→ 1:	#n_B
	2:	"*string*"
1: *n*	→ 1:	1 or 0
2: *counter starting value*		
1: *counter ending value*	→	
1: *counter incr. value*	→	
1: *obj*	→	
1: *time limit* (seconds)	→	
2: *obj*		
1: '*name*'	→	
1: { *date time* obj_{action} *int* }	→ 1:	*alarm index number*
1: { #$flags_{system}$ #$flags_{user}$ }	→	
1: { obj_1 key_1 obj_2 key_2... }	→	
2: *obj* or '*name*'		
1: '*name*' or *obj*	→	
2: *obj* or '*name*'		
1: '*name*' or *obj*	→	
2: *obj* or '*name*'		
1: '*name*' or *obj*	→	
2: *obj* or '*name*'		
1: '*name*' or *obj*	→	
1: *obj*	→	

Name (keyboard access)	Description
STREAM (PRG LIST PROC STREA)	Evaluate *obj* repeatedly, using as argument pairs each successive list *element* & previous result (first time use ele_1 and ele_2).
STR→ (must be typed)	Functions as OBJ→ on a string.
→STR (PRG TYPE →STR)	Convert *obj* to string.
STWS (MTH BASE NXT STWS)	Set *binary wordsize*.
SUB (◁ CHARS SUB)	Extract portion, *p*, of *list*, *string* or *grob*, as indicated by *start* and *end* positions.
SVD (MTH MATR FACTR SVD)	Compute singular value decomp. of *matrix A*, such that $A = U \times \mathrm{diag}(S) \times V$, where S contains the singular values of A.
SVL (MTH MATR FACTR NXT SVL)	*Matrix* sing. vals.
SWAP (◁ SWAP)	Exchange contents of first two levels in the stack.
SYSEVAL (must be typed)	Eval. obj. at system *address*.
%T (MTH REAL %T)	Compute the percentage of x represented by y.
→TAG (PRG TYPE →TAG)	Create tagged object from given *obj* and given *tag*, *name*, or value (x).
TAIL (◁ CHARS NXT TAIL)	Cut first elem. (str. or list).
TAN (TAN)	Tangent of argument.
TANH (MTH HYP TANH)	Hyperbolic tangent of arg.
TAYLR (◁ SYMBOLIC TAYLR)	Calculate *n*th order Taylor's polynomial in *name* for given *expression*.
TDELTA (◁ EQ LIB UTILS NXT TDELT)	Calculate change (difference) between given temperatures.
TEACH (must be typed)	Create EXAMPLES subdirectory.
TEXT (PRG NXT OUT TEXT)	Display stack display.
THEN (PRG BRCH IF THEN)	Begin true clause of IF structure (if argument is *true*).
TICKS (◁ TIME TICKS)	*System time* (8192 = 1 sec).
TIME (◁ TIME TIME)	*System time* (*HH.MMSSs*).
→TIME (◁ TIME →TIM)	Set system time (*HH.MMSSs*).
TINC (◁ EQ LIB UTILS NXT TINC)	Find a new temperature given initial temperature and change.
TLINE (PRG PICT TLINE)	Toggle every pixel in PICT along line between given coordinates.
TMENU (◁ MODES MENU TMEN)	Display given menu (but do not change contents of CST).
TOT (◁ STAT 1VAR TOT)	Sum ΣDAT columns.
TRACE (MTH MATR NORM NXT TRACE)	Trace of *matr*.
TRANSIO (◁ I/O IOPAR TRAN)	I/O char. *transl. option*.
TRN (MTH MATR MAKE TRN)	Transpose of *matrix*.
TRNC (MTH REAL NXT NXT TRNC)	Truncate to n dec. pl. ($1 \le n \le 11$), n sig. fig. ($n < 0$), or curr. disp. mode ($n > 11$).
TRUTH (◁ PLOT PTYPE TRUTH)	Set plot type to Truth.

Stack Inputs	→	Stack Outputs
2: { $ele_1\ ele_2\ ele_3 \ldots$ }		
1: obj	→ 1:	$result$
1: "obj"	→ 1:	(result of evaluation)
1: obj	→ 1:	"obj"
1: $binary\ wordsize$	→	
3: { $list$ } or "str" or $grob$		
2: $start$ or $(x, y)_{end}^{start}$		
1: end or $(x, y)_{end}^{start}$	→ 1:	{ p } or "p" or $grob_p$
	3:	[[U]]
	2:	[[V]]
1: [[A]]	→ 1:	[[S]]
1: [[$matrix$]]	→ 1:	[$vector$]$_{singular\ values}$
2: obj_A	2:	obj_B
1: obj_B	→ 1:	obj_A
1: #$address$	→	
2: x		
1: y	→ 1:	$100y/x$
2: obj	1:	:tag:obj or
1: "tag" or '$name$' or x	→	:$name$:obj or :x:obj
1: { $ele_1\ ele_2\ ele_3 \ldots$ }	→ 1:	{ $ele_2\ ele_3 \ldots$ }
1: z	→ 1:	tan(z)
1: z	→ 1:	tanh(z)
3: '$expression$'		
2: '$name$'		
1: n	→ 1:	'$expression_{Taylor}$'
2: T_{final}		
1: $T_{initial}$	→ 1:	ΔT

1: t/f	→ 1:	(depends)
	→ 1:	#$system\ time$ (ticks)
	→ 1:	$system\ time_{HH.MMSSs}$
1: $time_{HH.MMSSs}$	→	
2: $T_{initial}$		
1: ΔT	→ 1:	T_{final}
2: (x_A, y_A)		
1: (x_B, y_B)	→	
1: { $menu_{user}$ } or $n_{sys.menu}$	→	
	→ 1:	[$sum_1\ sum_2 \ldots sum_m$]
1: [[$matrix_{square}$]]	→ 1:	$trace$
1: $translation\ opt.$ (0-3)	→	
1: [[$matrix_A$ (n x m)]]	→ 1:	[[$matrix_B$ (m x n)]]
2: $value_A$		
1: n	→ 1:	$value_B$

Name (keyboard access)	Description
TSTR (◀ TIME NXT NXT TSTR)	Create a string from given *time* and *date*; include corres. day of week (*dow*).
TVARS (◀ MEMORY TVARS)	List all variables (in current directory) of given object *type(s)*.
TVM (◀ SOLVE TVM SOLVR)	Displ. TVM Solver menu.
TVMBEG (◀ SOLVE TVM BEG)	Set TVM Begin mode.
TVMEND (◀ SOLVE TVM END)	Set TVM End mode.
TVMROOT (◀ SOLVE TVM END)	Solve for given var.
TYPE (PRG TEST NXT TYPE)	Return *type* of *obj*.
UBASE (◀ UNITS UBASE)	Convert argument to SI units.
UFACT (◀ UNITS UFACT)	Re-express argument $unit_A$ to include given factor $unit_B$.
→UNIT (◀ UNITS UFACT)	Create unit object by combining x and *unit* portion of y.
UNTIL (PRG BRCH DO UNTIL)	Begin test clause of DO...UNTIL loop.
UPDIR (◀ UP)	Move up one directory level.
UTPC (MTH NXT PROB NXT UTPC)	Probability of exceeding x with randomly chosen value in chi-square distribution with n degrees of freedom.
UTPF (MTH NXT PROB NXT UTPF)	Prob. of exceeding x with randomly chosen value in Snedecor's F distribution, with n_1 and n_2 as num. and denom. deg. of freedom.
UTPN (MTH NXT PROB NXT UTPN)	Probability of exceeding x with randomly chosen value in normal distribution with mean m and variance v.
UTPT (MTH NXT PROB NXT UTPT)	Probability of exceeding x with randomly chosen value in Student's T distribution with n deg. of freedom.
UVAL (◀ UNITS UVAL)	Num. part of given unit object.
VAR (◀ STAT 1VAR NXT VAR)	Compute sample variance for each column in ΣDAT.
VARS (◀ MEMORY DIR VARS)	List all vars. (curr. dir).
VERSION (must be typed)	Show software version number and copyright message of HP 48G/GX.
VTYPE (PRG TYPE NXT NXT VTYPE)	Obj. type in *name*.
→V2 (MTH VECTR →V2)	Convert given two values to complex number or 2D vector (in current coord. mode).
→V3 (MTH VECTR →V3)	Convert given three values to 3D vector (in current coordinate mode—Rectangular, Cylindrical, or Spherical).
V→ (MTH VECTR V→)	Decompose a vector or complex number into its constituent elements; put these, in order, on the stack. For arguments with less than four elements, results will reflect current coordinate mode.
*W (◀ PLOT PPAR NXT *W)	Widen horiz. plt. scale.
WAIT (PRG NXT IN WAIT)	Suspend program execution for given interval or until key is pressed.

Stack Inputs	→	Stack Outputs
2:	*date*	
1:	*time* → 1:	"*dow date time*"
1:	{ *type(s)* } → 1:	{ $name_1$ $name_2$ … }
1:	'$name_{TVM}$' → 1:	$value_{TVM}$
1:	*obj* → 1:	*object type*
1:	*x_unit* → 1:	*y_SI-unit*
2:	x_unit_A	
1:	y_unit_B → 1:	$x\_unit_B * unit_C$
2:	*x*	
1:	*y_unit* → 1:	*x_unit*
2:	*n*	
1:	*x* → 1:	UTPC(*n*,*x*)
3:	n_1	
2:	n_2	
1:	*x* → 1:	UTPF(n_1,n_2,*x*)
3:	*m*	
2:	*v*	
1:	*x* → 1:	UTPN(*m*,*v*,*x*)
2:	*n*	
1:	*x* → 1:	UTPC(*n*,*x*)
1:	*x_unit* → 1:	*x*
	→ 1:	[var_1 var_2 …]
	→ 1:	{ $name_1$ $name_2$ … }
	2:	"*software version no.*"
	→ 1:	"*copyright message*"
1:	'*name*' → 1:	*object type*
2:	*x*	
1:	*y* → 1:	[*x y*] or (*x*, *y*)
3:	*a*	
2:	*b*	
1:	*c* → 1:	[*a b c*]
	n:	ele_1
	:	
	2:	ele_{n-1}
1:	[ele_1 ele_2 … ele_n] → 1:	ele_n
1:	*widening factor* →	
1:	*interval* (seconds) or 0 → 1: (or −1 for curr. menu)	(nothing) or *keycode*

Name (keyboard access)	Description
WHILE (PRG BRCH WHILE WHILE)	Begin WHILE loop.
WIREFRAME (↶PLOT NXT 3D PTYPE WIREF)	Set plot type to Wireframe.
WSLOG (must be typed)	Return strings showing four most recent warmstart *events*. Each *event* is of the form "*code*–*date time*", where *code* is one hex digit denoting the reason for, or nature of, the *event*.
ΣX (↶STAT SUMS ΣX)	Sum indep. var. in ΣDAT.
ΣX^2 (↶STAT SUMS ΣX^2)	Sum (ind. var.)² in ΣDAT.
XCOL (↶STAT ΣPAR XCOL)	Identify ΣDAT ind. var.
XMIT (↶I/O NXT SERIA XMIT)	Send data string serially (no Kermit); return unsent portion and success *t/f*.
XOR (PRG TEST NXT XOR)	Compute Exclusive OR of given arguments.
XPON (MTH REAL NXT XPON)	Exponent of arg. value.
XRECV (↶I/O NXT XRECV)	Receive via XModem; store.
XRNG (↶PLOT PPAR XRNG)	Specify *x*-axis display range in PPAR.
XROOT (↷X√y)	Calculate the "*x*th" root of the given argument, *y*.
XSEND (↶I/O NXT XSEN)	Send copy via XModem.
XVOL (↶PLOT NXT 3D VPAR XVOL)	Specify width of view volume in VPAR.
XXRNG (↶PLOT NXT 3D VPAR XXRN)	Specify *x*-range of input plane in VPAR.
ΣX*Y (↶STAT SUMS ΣX*Y)	Sum (independent var. × dependent var.) in ΣDAT.
ΣY (↶STAT SUMS ΣY)	Sum depend. var. in ΣDAT.
ΣY^2 (↶STAT SUMS ΣY^2)	Sum (dep. var.)² in ΣDAT.
YCOL (↶STAT ΣPAR YCOL)	Identify ΣDAT dep. var.
YRNG (↶PLOT PPAR YRNG)	Specify *y*-axis display range in PPAR.
YSLICE (↶PLOT NXT 3D PTYPE YSLIC)	Set Plot Type to Y-Slice.
YVOL (↶PLOT NXT 3D VPAR YVOL)	Specify depth of view volume in VPAR.
YYRNG (↶PLOT NXT 3D VPAR YYRN)	Specify *y*-range of input plane in VPAR.
ZFACTOR (↶EQ LIB UTILS ZFACT)	*Compressibility* of non-ideal, hydrocarbon gas at *reduced temp.* & *pressure*.
ZVOL (↶PLOT NXT 3D VPAR ZVOL)	Specify height of view volume in VPAR.
+ (+)	Add the two given arguments (also valid for combining arguments of many types).
- (−)	Find the difference of the two given arguments (also valid for combining arguments of many types).
* (×)	Multiply the two given arguments (also valid for combining arguments of many types).

Stack Inputs	→	Stack Outputs
	4:	"$event_4$"
	3:	"$event_3$"
	2:	"$event_2$"
	→ 1:	"$event_1$"
	→ 1:	Σx
	→ 1:	Σx^2
1: $column_{independent}$	→	
	2:	("*unsent substring*")
1: "*string*"	→ 1:	1 or 0
2: t/f_A or $\#n_A$		
1: t/f_B or $\#n_B$	→ 1:	1 or 0
1: x	→ 1:	$n_{exponent}$
1: '*name*'	→	
2: x_{min}		
1: x_{max}	→	
2: y		
1: x	→ 1:	$\sqrt[x]{y}$
1: '*name*'	→	
2: *left*		
1: *right*	→	
2: x_{min}		
1: x_{max}	→	
	→ 1:	Σxy
	→ 1:	Σy
	→ 1:	Σx^2
1: $column_{dependent}$	→	
2: y_{min}		
1: y_{max}	→	
2: *near*		
1: *far*	→	
2: y_{min}		
1: y_{max}	→	
2: $temperature_{reduced}$		
1: $pressure_{reduced}$	→ 1:	*compression (z-) factor*
2: *low*		
1: *high*	→	
2: z_A		
1: z_B	→ 1:	$z_A + z_B$
2: z_A		
1: z_B	→ 1:	$z_A - z_B$
2: z_A		
1: z_B	→ 1:	$z_A z_B$

Name (keyboard access)	Description
/ (÷)	Find quotient of the two given arguments (also valid for combining arguments of many types).
^ (y^x)	Raise the first argument to the power of the second argument.
< (PRG TEST <)	Test if first argument, *x*, is less than second argument, *y*.
≤ (PRG TEST ≤)	Test if first argument, *x*, is less than or equal to second argument, *y*.
> (PRG TEST >)	Test if first argument, *x*, is greater than second argument, *y*.
≥ (PRG TEST ≥)	Test if first argument, *x*, is greater than or equal to second argument, *y*.
= (←=)	Equate two arguments in a symbolic expression.
== (PRG TEST ==)	Test if first argument, *x*, is equal to second argument, *y*.
≠ (PRG TEST ≠)	Test if first argument, *x*, is not equal to second argument, *y*.
! (MTH NXT PROB !)	Factorial or Gamma func.
∫ (→∫)	Integrate given *integrand* with respect to *named* variable from $limit_{lower}$ to $limit_{upper}$. Results may be either numeric or symbolic, depending on the state of Flag -3.
∂ (→∂)	Differentiate given *expression* with respect to *named* variable.
% (MTH REAL %)	Compute *x* percent of *y*.
π (←π)	Return 'π' or its numeric equivalent.
Σ (←Σ)	Compute the sum of *summand* for *index* values ranging from *start* through *finish* (in increments of one).
Σ+ (←STAT DATA Σ+)	Add 1 or more data points to ΣDAT. Use multiple stack levels for more than one point of 1-value data; use a vector for one point (and a matrix for more than one point) of multi-value data.
Σ- (←STAT DATA Σ-)	Last values in ΣDAT via Σ+.
√ (←√x)	Square root of argument.
_ (→_)	Attach given *unit* to given real number.
| (←SYMBOLIC NXT |)	Substitute the given *v*alues (numeric or symbolic) for the given *n*ames in *expression*.
→ (→→)	Bind given *obj*ects, in order, to the respective local names supplied following this command.

Stack Inputs	→	Stack Outputs
2: z_A		
1: z_B	→ 1:	$z_A \div z_B$
2: w		
1: z	→ 1:	w^z
2: x		
1: y	→ 1:	0 or 1
2: x		
1: y	→ 1:	0 or 1
2: x		
1: y	→ 1:	0 or 1
2: x		
1: y	→ 1:	0 or 1
2: z_A		
1: z_B	→ 1:	'z_A=z_B'
2: x		
1: y	→ 1:	0 or 1
2: x		
1: y	→ 1:	0 or 1
1: n or x	→ 1:	$n!$ or $\Gamma(x+1)$
4: $limit_{lower}$		
3: $limit_{upper}$		
2: '*integrand*'		
1: '*name*'	→ 1:	*integral* or '*integral*'
2: '*expression*'		
1: '*name*'	→ 1:	'*derivative*'
2: x		
1: y	→ 1:	$xy/100$
	→ 1:	'π' or 3.14159265359
4: '*index*'		
3: *start*		
2: *finish*		
1: '*summand*'	→ 1:	*sum* or '$expr_{sum}$'
m: x_1 (or nothing)		
:		
2: x_{m-1} (or nothing)		
1: x or x_m or [x_1 x_2 … x_m]	→	
	→ 1:	[x_1 x_2 … x_m]
1: z	→ 1:	$\sqrt{z}$
2: x		
1: '*unit*'	→ 1:	*x_unit*
2: '$expression_A$'		
1: { n_1 'v_1' n_2 'v_2' … }	→ 1:	'$expression_B$'
n: obj_1		
:		
2: obj_{n-1}		
1: obj_n	→	

SYSTEM FLAGS (• = *default*)

Flag		Description of States
- 1	Set:	QUAD and ISOL return all possible sols.
	• Clr:	QUAD and ISOL return principle solution.
- 2	Set:	Symb. const. stay symb. (if Flag -3 is clr).
	• Clr:	Symbolic constants eval. to numbers.
- 3	Set:	All symbolic arguments stay symbolic.
	• Clr:	All symbolic arguments eval. to numbers.
- 4	Set:	*(unused)*
	• Clr:	*(unused)*
- 5	• Set:	6th bit (value 32) of binary wordsize is 1.*
	Clr:	6th bit (value 32) of binary wordsize is 0.*
- 6	• Set:	5th bit (value 16) of binary wordsize is 1.*
	Clr:	5th bit (value 16) of binary wordsize is 0.*
- 7	• Set:	4th bit (value 8) of binary wordsize is 1.*
	Clr:	4th bit (value 8) of binary wordsize is 0.*
- 8	• Set:	3rd bit (value 4) of binary wordsize is 1.*
	Clr:	3rd bit (value 4) of binary wordsize is 0.*
- 9	• Set:	2nd bit (value 2) of binary wordsize is 1.*
	Clr:	2nd bit (value 2) of binary wordsize is 0.*
-10	• Set:	1st bit (value 1) of binary wordsize is 1.*
	Clr:	1st bit (value 1) of binary wordsize is 0.*
-11	Set:	HEX or OCT mode (dep. on Flag -12).
	• Clr:	DEC or BIN mode (dep. on Flag -12).
-12	Set:	HEX or BIN mode (dep. on Flag -11).
	• Clr:	OCT or DEC mode (dep. on Flag -11).
-13	Set:	*(unused)*
	• Clr:	*(unused)*
-14	Set:	TVM calcs. use BEGIN payment mode.
	• Clr:	TVM calcs. use END payment mode.
-15	Set:	Spherical coord. mode (if Flag -16 is set).
	• Clr:	Cylindr. coord. mode (if Flag -16 is set).
-16	Set:	Polar coordinate mode.
	• Clr:	Rectangular coordinate mode.
-17	Set:	Radians angle mode.
	• Clr:	Not Radians angle mode (see Flag -18).
-18	Set:	Grads angle mode (if Flag -17 is clear).
	• Clr:	Degrees angle mode (if Flag -17 is clear).
-19	Set:	→V2 creates complex number.
	• Clr:	→V2 creates 2-D vector.
-20	Set:	Underflow treated as error.
	• Clr:	Underflow returns 0; sets Flag -23 or -24.
-21	Set:	Overflow treated as error.
	• Clr:	Overflow sets Flag -25 and returns (±)9.99999999999E499.

**The system binary wordsize is defined as a value 1 greater than the sum of the bit values of Flags -5 to -10.*

SYSTEM FLAGS (• = *default*)

Flag		Description of States
-22	Set:	Infinite result sets Flag -26 and returns (±)9.99999999999E499.
	• Clr:	Infinite result treated as error.
-23	Set:	Negative underflow condition exists.*
	• Clr:	No negative underflow condition exists.
-24	Set:	Positive underflow condition exists.*
	• Clr:	No positive underflow condition exists.
-25	Set:	Overflow condition exists.*
	• Clr:	No overflow condition exists.
-26	Set:	Infinite result condition exists.*
	• Clr:	No infinite result condition exists.
-27	Set:	Display symbolic complex as 'x+y*i'.
	• Clr:	Display symbolic complex as '(x,y)'.
-28	Set:	Plot multiple equations simultaneously.
	• Clr:	Plot multiple equations serially.
-29	Set:	No axes drawn for 2-D and stat. plots.
	• Clr:	Axes drawn for 2-D and stat. plots.
-30	Set:	*(unused)*
	• Clr:	*(unused)*
-31	Set:	No curve filling between plotted points.
	• Clr:	Curve filling between plotted points.
-32	Set:	Graphics cursor inverse of background.
	• Clr:	Graphics cursor always dark.
-33	Set:	I/O sent to IR (infrared) port.
	• Clr:	I/O sent to serial port.
-34	Set:	Print output to serial port (if Flag -33 clr.).
	• Clr:	Print output to IR (infrared) printer.
-35	Set:	I/O objects sent in binary form.
	• Clr:	I/O objects sent in ASCII form.
-36	Set:	Matching name overwrites in I/O receive.
	• Clr:	Matching name changes in I/O receive.
-37	Set:	Double-spaced printing.
	• Clr:	Single-spaced printing.
-38	Set:	No linefeed added after each print line.
	• Clr:	Linefeed added after each print line.
-39	Set:	I/O messages suppressed.
	• Clr:	I/O messages displayed.
-40	Set:	Clock always displayed.
	• Clr:	Clock displayed only with TIME menu.
-41	Set:	24-hour clock format.
	• Clr:	12-hour clock format.
-42	Set:	DD.MM.YY date format.
	• Clr:	MM/DD/YY date format.

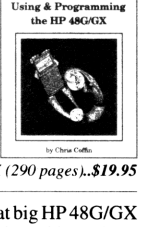
**This flag is set only if condition is not treated as error.*

SYSTEM FLAGS (• = *default*)

Flag		Description of States
-43	Set:	Unackn. repeat alarms not rescheduled.
	• Clr:	Unackn. repeat alarms rescheduled.
-44	Set:	Ackn. alarms retained in alarm list.
	• Clr:	Ackn. alarms deleted from alarm list.
-45	Set:	1st bit (value 1) of displ.-digit count is 1.
	• Clr:	1st bit (value 1) of displ.-digit count is 0.
-46	Set:	2nd bit (value 2) of displ.-digit count is 1.
	• Clr:	2nd bit (value 2) of displ.-digit count is 0.
-47	Set:	3rd bit (value 4) of displ.-digit count is 1.
	• Clr:	3rd bit (value 4) of displ.-digit count is 0.
-48	Set:	4th bit (value 8) of displ.-digit count is 1.
	• Clr:	4th bit (value 8) of displ.-digit count is 0.
-49	Set:	FIX or ENG mode (dep. on Flag -50).
	• Clr:	STD or SCI mode (dep. on Flag -50).
-50	Set:	SCI or ENG mode (dep. on Flag -49).
	• Clr:	FIX or STD mode (dep. on Flag -49).
-51	Set:	Fraction mark is , (comma).
	• Clr:	Fraction mark is . (period).
-52	Set:	Level-1 object displayed on 1 line.
	• Clr:	Level-1 object displayed on up to 4 lines.
-53	Set:	Display all parentheses in alg. expr.
	• Clr:	Suppress some parentheses in alg. expr.
-54	Set:	Relatively small singular matrix values not converted to 0; DET does not round.
	• Clr:	Relatively small singular mat. values converted to 0; DET rounds automatically.
-55	Set:	Save arguments of last command.
	• Clr:	Do not save arguments of last command.
-56	Set:	BEEP and error tones disabled.
	• Clr:	BEEP and error tones enabled.
-57	Set:	Alarm tone disabled.
	• Clr:	Alarm tone enabled.
-58	Set:	Do not display parameter variable data.
	• Clr:	Display parameter variable data.
-59	Set:	Variable Browser shows names only.
	• Clr:	Variable Browser shows names/contents.
-60	Set:	Press [α] once for alpha lock.
	• Clr:	Press [α] twice for alpha lock.
-61	Set:	Press [←][USER] once for user mode lock.
	• Clr:	Press [←][USER] twice for user mode lock.
-62	Set:	User mode on.
	• Clr:	User mode off.
-63	Set:	User-defined [ENTER] activated.
	• Clr:	[ENTER] evaluated command line.
-64	Set:	Last GETI or PUTI wrapped index (to 1).
	• Clr:	Last GETI or PUTI did not wrap index.

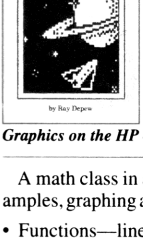
Other Books from Grapevine Publications

If you liked this Pocket Guide, here are some other Grapevine books that may be helpful, too:



Quickly learn problem-solving with matrices, units, complex numbers, vectors, and algebraic objects—using clear examples and explanations. Next, study program basics, with loops, branches, flags, etc. Finally, program with built-in applications and design your own—with directories and custom menus, too.

Easy Course Using/Programming HP 48G/GX *(290 pages)*..**$19.95**

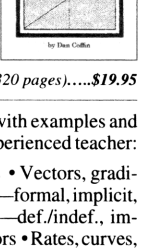


Use the power of that big HP 48G/GX display. Learn to build graphics objects ("grobs") and to use them to customize displays with diagrams, pictures, and data plots. You get a great review of the built-in graphics tools, also. Then you're ready to build your own grobs and use them in your own programs.

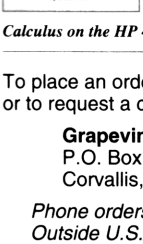
Graphics on the HP 48G/GX *(300 pages)*.....**$19.95**

A math class in a book—lessons, examples, graphing and problem-solving:

• Functions—linear, quadratic, rational, polynomial • Trig • Geometry • Conics • Equations of lines & planes • Inequalities • Vectors • Programs for plotting and solving • Written by an experienced classroom math teacher.



Algebra & Pre-Calculus on the HP 48G/GX *(320 pages)*.....**$19.95**



Start college now, with examples and programs from an experienced teacher:

• Limits, sums, series • Vectors, gradients • Differentiation—formal, implicit, partial • Integration—def./indef., improper, by parts, vectors • Rates, curves, constraints, growth/decay, force/velocity/accel., surfaces/solids, and more.

Calculus on the HP 48G/GX *(330 pages)*.....**$19.95**

To place an order with either Visa or MasterCard, or to request a complete catalog, contact:

Grapevine Publications, Inc.
P.O. Box 2449
Corvallis, OR 97339-2449 U.S.A.

Phone orders/catalog req.:	(800) 338-4331
Outside U.S./Canada:	(541) 754-0583
Fax:	(541) 754-6508

Operator Precedence

In an algebraic expression, operators have these relative priorities (and = is last of all): ---> Same-priority operations in an expression proceed *left to right*.

1 ( )	6 + and −
2 Functions	7 <, >, ==, etc.
3 ! (Factorial)	8 AND and NOT
4 ^ and √	9 OR and XOR
5 *, /, − (NEG)	10 | ("where")

Index

◀ and ▶ (shift keys) 2-3
α 2-3
Alarms 7, 31
Algebraic objects 1, 20, *above*
Alpha characters . 3, 7, 10-11
Alpha keyboard ..... *cover*
Angle 11
Annunciators 3
Arguments 5
Arithmetic 5, 9
Arrow keys 2, 4, 9, 20, 22
Backups 34-35
Batteries 39
Binary numbers 1, 18-19
Branching 28-29
CANCEL 2-4, 11
CHOOS 21
Clearing 3, 6
Command line 2-4
Commands 41-77
Common commands 3
Common problems 40-41
Complex numbers ... 1, 18-19
Conic plots 14
Constants 7, 18-19, 37
Copying and moving 9-10
Curve-fitting (regression) ... 27
Custom menus 7, 38
∂ and Differentiation ... 23-24
Deleting 2
Delimiters 2
Derivatives 13, 23-24
Directories 1, 8-10
Display 11, 39
Editing 2-3, 20, 26
ENTER 3, 12
Errors and recovery 4, 30
EQ LIB 36-37
EQUATION & EquationWriter 20
EVAL 21, 23-25
Expressions ..... 20-21, *above*
Flags 11, 21, 26, 78-80
Functions 1, 12-14, 22, 24
Graphics environment ... 7, 13
Graphics objects (grobs) ... 30
HOME 7-8
I/O and data comm. 32-33
Inequalities 13
Inputs and outputs 28, 30
Integration 24-25
Interactive stack 4
Isolating a variable 22
Key assignments 38-39
Keys 2-3, 38-39
Libraries 1, 34-35
Lists 1, 6, 18-19, 29
Looping 28-29
Matrices . 1, 18-19, 26, 27, 29
Memory 8-9
Menus (typing aids) 6, 10
MODES 11
MTH 18-19
Mult. Eq. Solver (MES) 36-37
Names 1, 6-7
NUM 25
Numeric results 21, 23, 26
Objects (and types) 1, 29
Operator precedence .. *above*
Parametric plots 14
Paths 8, 10
PICT 7, 30
PLOT and plotting 7, 12-14
Polar plots 14
Ports 34-35, 39
Postfix notation 5
PRG and programs .. 1, 28-30
Principal/general sol. ... 21-22
Printing 7, 32-33
Probability 18-19
Purging 6, 9
Quadratic equations 21
RAD 11, 40
Real numbers 1, 18-19
Recalling values 6, 15
Reserved names 7
Resetting 31, 39
Roots of functions 13, 15
Σ 26
Scaling and viewing 12-14
Simultaneous equations 15
SOLVE and solving 7, 15
SPC 2
STACK 2-4
STAT and statistics ... 7, 26-27
STO & storing values . 6-7, 15
Strings 1, 11
Summation 7, 26
SYMBOLIC 21-25
Symbolic results 21, 23, 26
System information 39
Tags 1
Taylor polynomials 22
TIME 31
Troubleshooting 40-41
UNDO 4
Units 1, 16-17
VAR 6, 10, 25
Variables 6, 22
Vectors 1, 18-19
Zooming 7, 12-13

ISBN 0-931011-45-0