When evaluated, *e* returns its numerical representation if flag −2 or flag −3 is set; otherwise, *e* returns its symbolic representation.

Remarks: The number returned for e is the closest approximation of the constant *e* to 12-digit accuracy. For exponentiation, use the expression 'EXP(X)' rather than 'e^X', since the function EXP uses a special algorithm to compute the exponential to greater accuracy.

Related Commands: EXP, EXPM, i, LN, LNP1, MAXR, MINR, π

EGV

Eigenvalues and Eigenvectors Command: Computes the eigenvalues and right eigenvectors for a square matrix.

{}

Level 1	→	Level 2	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_{EVec}$	[*vector*]$_{EVal}$

Keyboard Access: (MTH) MATR (NXT) EGV

Affected by Flags: None

Remarks: The resulting vector EVal contains the computed eigenvalues. The columns of matrix EVec contain the right eigenvectors corresponding to the elements of vector EVal.

The computed results should minimize (within computational precision):

$$\frac{|A \cdot EVec - EVec \cdot \mathrm{diag}(EVal)|}{n \cdot |A|}$$

where diag($EVal$) denotes the $n \times n$ diagonal matrix containing the eigenvalues $EVal$.

Related Commands: EGVL

EGVL

Eigenvalues Command: Computes the eigenvalues of a square matrix.

{}

Level 1	→	Level 1
[[*matrix*]]$_A$	→	[*vector*]$_{EVal}$

Keyboard Access: (MTH) MATR (NXT) EGVL

Affected by Flags: None

Remarks: The resulting vector L contains the computed eigenvalues.

Related Commands: EGV

ELSE

ELSE Command: Starts false clause in conditional or error-trapping structure.

See the IF and IFERR keyword entries for syntax information.

Keyboard Access:

(PRG) BRCH IF ELSE

(PRG) (NXT) ERROR IFERR ELSE

Remarks: See the IF and IFERR keyword entries for more information.

Related Commands: IF, IFERR, THEN, END

END

END Command: Ends conditional, error-trapping, and indefinite loop structures.

See the IF, CASE, IFERR, DO, and WHILE keyword entries for syntax information.

Keyboard Access:

PRG BRCH IF END

PRG BRCH CASE END

PRG BRCH DO END

PRG BRCH WHILE END

PRG NXT ERROR IFERR END

Remarks: See the IF, CASE, IFERR, DO, and WHILE keyword entries for more information.

Related Commands: IF, CASE, DO, ELSE, IFERR, REPEAT, THEN, UNTIL, WHILE,

ENDSUB

Ending Sublist Command: Provides a way to access the total number of sublists contained in the list used by DOSUBS.

Keyboard Access: PRG LIST PROC ENDS

Affected by Flags: None

Remarks: Returns an Undefined Local Name error if executed when DOSUBS is not active.

Example: The following program subtracts the number of elements in a list from each element in the list:

```
« → a
  « a 1
    « ENDSUB -
  »
» DOSUBS
»
```

Related Commands: DOSUBS, NSUB

ENG

Engineering Mode Command: Sets the number display format to Engineering mode, which displays one to three digits to the left of the fraction mark (decimal point) and an exponent that is a multiple of three. The total number of significant digits displayed is $n + 1$.

{}

Level 1	→	Level 1
n	→	

Keyboard Access: ◀ MODES FMT ENG

Affected by Flags: None

Remarks: Engineering mode uses $n + 1$ significant digits, where $0 \leq n \leq 11$. (Values for n outside this range are rounded up or down.) A number is displayed or printed as follows:

(*sign*) *mantissa* E (*sign*) *exponent*

where the mantissa is of the form $(nn)n.(n \ldots)$ (with up to 12 digits total) and the exponent has one to three digits.

A number with an exponent of −499 is displayed automatically in Scientific mode.

Example: The number 103.6 in Engineering mode with five significant digits (n=4) would appear as `103.60E0`. This same number with one significant digit (n=0) would appear as `100.E0`.

Related Commands: FIX, SCI, STD

EQ→

Equation to Stack Command: Separates an equation into its left and right sides.

{ }

Level 1	→	Level 2	Level 1
'$symb_1 = symb_2$'	→	'$symb_1$'	'$symb_2$'
z	→	z	0
'name'	→	'name'	0
x_unit	→	x_unit	0
'symb'	→	'symb'	0

Keyboard Access: PRG TYPE NXT EQ→

Affected by Flags: None

Remarks: If the argument is an expression, it is treated as an equation whose right side equals zero.

Related Commands: ARRY→, DTAG, LIST→, OBJ→, STR→

EQNLIB

Equation Library Command: Starts the Equation Library application.

Keyboard Access: ← EQ LIB EQ LIB EQNLIB

Affected by Flags: None

Remarks: The Equation Library is a collection of equations and commands useful for solving typical science and engineering problems.

Related Commands: MSOLVR, SOLVEQN

ERASE

Erase PICT Command: Erases *PICT*, leaving a blank *PICT* of the same dimensions.

Keyboard Access:

← PICTURE ERASE NXT ERASE

← PICTURE ← CLEAR

← PLOT ERASE

Affected by Flags: None

Related Commands: DRAW

ERRM

Error Message Command: Returns a string containing the error message of the most recent calculator error.

Level 1	→	Level 1
	→	"error message"

Keyboard Access: PRG NXT ERROR ERRM

Affected by Flags: None

Remarks: ERRM returns the string for an error generated by DOERR. If the argument to DOERR was 0, the string returned by ERRM is empty.

Example: The program « IFERR + THEN ERRM END » returns "Bad Argument Type" to level 1 if improper arguments (for example, a complex number and a binary integer) are in levels 1 and 2.

Related Commands: DOERR, ERRN, ERR0

ERRN

Error Number Command: Returns the error number of the most recent calculator error.

Level 1	→	Level 1
	→	#n_{error}

Keyboard Access: PRG NXT ERROR ERRN

Affected by Flags: None

Remarks: If the most recent error was generated by DOERR with a string argument, ERRN returns # 70000h. If the most recent error was generated by DOERR with a binary integer argument, ERRN returns that binary integer. (If the most recent error was generated by DOERR with a real number argument, ERRN returns the binary integer conversion of the real number.)

Example: The program « IFERR + THEN ERRN END » returns # 202h to level 1 if improper arguments (for, example, a complex number and a binary integer) are in levels 1 and 2.

Related Commands: DOERR, ERRM, ERR0

ERR0

Clear Last Error Number Command: Clears the last error number so that a subsequent execution of ERRN returns # 0h, and clears the last error message.

Keyboard Access: PRG NXT ERROR ERR0

Affected by Flags: None

Related Commands: DOERR, ERRM, ERRN

EVAL

Evaluate Object Command: Evaluates the object.

Level 1	→	Level 1
obj	→	*(see below)*

Keyboard Access: EVAL

Affected by Flags: Numerical Results (−3)

Remarks: The following table describes the effect of the evaluation on different object types.

Obj. Type	Effect of Evaluation
Local Name	Recalls the contents of the variable.
Global Name	*Calls* the contents of the variable: ■ A name is evaluated. ■ A program is evaluated. ■ A directory becomes the current directory. ■ Other objects are put on the stack. If no variable exists for a given name, evaluating the name returns the name to the stack.
Program	*Enters* each object in the program: ■ Names are evaluated (unless quoted). ■ Commands are evaluated. ■ Other objects are put on the stack.
List	*Enters* each object in the list: ■ Names are evaluated. ■ Commands are evaluated. ■ Programs are evaluated. ■ Other objects are put on the stack.

Obj. Type	Effect of Evaluation
Tagged	If the tag specifies a port, recalls and evaluates the specified object. Otherwise, puts the untagged object on the stack.
Algebraic	*Enters* each object in the algebraic expression: ■ Names are evaluated. ■ Commands are evaluated. ■ Other objects are put on the stack.
Command, Function, XLIB Name	Evaluates the specified object.
Other Objects	Puts the object on the stack.

To evaluate a symbolic argument to a numerical result, evaluate the argument in Numerical Result mode (flag −3 set) or execute →NUM on that function.

Related Commands: →NUM, SYSEVAL

EXP

Exponential Analytic Function: Returns the exponential, or natural antilogarithm, of the argument; that is, e raised to the given power.

{}

Level 1	→	Level 1
z	→	e^z
'symb'	→	'EXP(symb)'

Keyboard Access: ◀ e^x

Affected by Flags: Numerical Results (−3)

Remarks: EXP uses extended precision constants and a special algorithm to compute its result to full 12-digit precision for all arguments that do not trigger and underflow or overflow error.

EXP provides a more accurate result for the exponential than can be obtained by using e y^x. The difference in accuracy increases as z increases. For example:

z	EXP(z)	e^z
3	20.0855369232	20.0855369232
10	22026.4657948	22026.4657949
100	2.68811714182E43	2.68811714191E43
500	1.40359221785E217	1.40359221809E217
1000	1.97007111402E434	1.97007111469E434

For complex arguments,

$$e^{(x,y)} = e^x cosy + ie^x siny$$

Related Commands: ALOG, EXPM, LN, LOG

EXPAN

Expand Products Command: Rewrites an algebraic expression or equation by expanding products and powers.

{}

Level 1	→	Level 1
'symb$_1$'	→	'symb$_2$'
x	→	x
(x, y)	→	(x, y)

Keyboard Access: ◀ SYMBOLIC EXPAN

Affected by Flags: None

Examples: 'A*(B+C)' EXPAN returns 'A*B+A*C'

EXPAN

'A^(B+C)' EXPAN returns 'A^B*A^C'

'X^5' EXPAN returns 'X*X^4'

'(X+Y)^2' EXPAN returns 'X^2+2*X*Y+Y^2'

Related Commands: COLCT, ISOL, QUAD, SHOW

EXPFIT

Exponential Curve Fit Command: Stores EXPFIT as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the exponential curve fitting model.

Keyboard Access: ◀ STAT ΣPAR MODL EXPF

Affected by Flags: None

Remarks: LINFIT is the default specification in ΣPAR. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, LR, LINFIT, LOGFIT, PWRFIT

EXPM

Exponential Minus 1 Analytic Function: Returns $e^x - 1$.

{}

Level 1	→	Level 1
x	→	$e^x - 1$
'symb'	→	'EXPM(symb)'

Keyboard Access: MTH HYP NXT EXPM

Affected by Flags: Numerical Results (−3)

Remarks: For values of x close to zero, 'EXPM(x)' returns a more accurate result than does 'EXP(x)-1' (Using EXPM allows

both the argument and the result to be near zero, and avoids an intermediate result near 1. The calculator can express numbers within 10^{-449} of zero, but within only 10^{-11} of 1.)

Related Commands: EXP, LNP1

EYEPT

Eye Point Command: Specifies the coordinates of the eye point in a perspective plot.

{}

Level 3	Level 2	Level 1	→	Level 1
x_{point}	y_{point}	z_{point}	→	

Keyboard Access: ◀ PLOT NXT 3D VPAR NXT EYEPT

Affected by Flags: None

Remarks: x_{point}, y_{point}, and z_{point} are real numbers that set the x-, y-, and z-coordinates as the eye-point from which to view a 3D plot's view volume. The y-coordinate must always be 1 unit less than the view volume's nearest point (y_{near} of YVOL). These coordinates are stored in the reserved variable $VPAR$.

Related Commands: NUMX, NUMY, XVOL, XXRNG, YVOL, YYRNG, ZVOL

F0λ

Black Body Emissive Power Function: Returns the fraction of total black-body emissive power.

{}

Level 2	Level 1	→	Level 1
y_{lambda}	x_T	→	x_{power}
y_{lambda}	'symb'	→	'F0λ(y_{lambda},symb)'
'symb'	x_T	→	'F0λ(symb,x_T)'
'$symb_1$'	'$symb_2$'	→	'F0λ($symb_1$,$symb_2$)'

Keyboard Access: ↩ EQ LIB [illegible]

Affected by Flags: Numerical Results (−3)

Remarks: F0λ calculates the fraction of total black-body emissive power at temperature x_T between wavelengths 0 and y_{lambda}. If units are not specified, y_{lambda} has implied units of meters and x_T has implied units of K.

F0λ returns a dimensionless fraction.

FACT

Factorial (Gamma) Function: Provided for compatibility with the HP 28. FACT is the same as !. See !.

{}

Level 1	→	Level 1
n	→	$n!$
x	→	$\Gamma(x+1)$
'symb'	→	'(symb)!'

Keyboard Access: None. Must be typed in.

FANNING

Fanning Friction Factor Function: Calculates the Fanning friction factor of certain fluid flows.

{}

Level 2	Level 1	→	Level 1
$x_{x/D}$	y_{Re}	→	$x_{fanning}$
$x_{x/D}$	'symb'	→	'FANNING($x_{x/D}$,symb)'
'symb'	y_{Re}	→	'FANNING(symb,y_{Re})'
'$symb_1$'	'$symb_2$'	→	'FANNING($symb_1$,$symb_2$)'

Keyboard Access: ↩ EQ LIB [illegible]

Affected by Flags: Numerical Results (−3)

Remarks: FANNING calculates the Fanning friction factor, a correction factor for the frictional effects of fluid flows having constant temperature, cross-section, velocity, and viscosity (a typical pipe flow, for example). $x_{x/D}$ is the relative roughness (the ratio of the conduit roughness to its diameter). y_{Re} is the Reynolds number. The function uses different computation routines for laminar flow ($Re \leq 2100$) and turbulent flow ($Re > 2100$). $x_{x/D}$ and y_{Re} must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Related Commands: DARCY

FC?

Flag Clear? Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set.

{}

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	0/1

Keyboard Access:

PRG TEST NXT NXT FC?

MODES FLAG FC?

Affected by Flags: None

Related Commands: CF, FC?C, FS?, FS?C, SF

FC?C

Flag Clear? Clear Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set. After testing, clears the flag.

{}

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	0/1

Keyboard Access:

PRG TEST NXT NXT FC?C

MODES FLAG FC?C

Affected by Flags: None

Example: If flag −44 is set, −44 FC?C returns 0 to level 1 and clears flag −44.

Related Commands: CF, FC?, FS?, FS?C, SF

FFT

Discrete Fourier Transform Command: Computes the one- or two-dimensional discrete Fourier transform of an array.

{}

Level 1	→	Level 1
[*array*]$_1$	→	[*array*]$_2$

Keyboard Access: MTH NXT FFT FFT

Affected by Flags: None

Remarks: If the argument is an N-vector or an $N\times1$ or $1\times N$ matrix, FFT computes the one-dimensional transform. If the argument is an $M\times N$ matrix, FFT computes the two-dimensional transform. M and N must be integral powers of 2.

The one-dimensional discrete Fourier transform of an N-vector X is the N-vector Y where:

$$Y_k = \sum_{n=0}^{N-1} X_n e^{-\frac{2\pi ikn}{N}} \qquad i = \sqrt{-1}$$

for $k = 0,\ 1,\ \ldots,\ N-1$.

The two-dimensional discrete Fourier transform of an $M \times N$ matrix X is the $M \times N$ matrix Y where:

$$Y_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{mn} e^{-\frac{2\pi ikm}{M}} e^{-\frac{2\pi iln}{N}} \qquad i = \sqrt{-1}$$

for $k = 0,\ 1,\ \ldots,\ M-1$ and $l = 0,\ 1,\ \ldots,\ N-1$.

FFT

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The FFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

Related Commands: IFFT

FINDALARM

Find Alarm Command: Returns the alarm index n_{index} of the first alarm due after the specified time.

Level 1	→	Level 1
date	→	n_{index}
{ *date time* }	→	n_{index}
0	→	n_{index}

Keyboard Access: ◀ TIME FCNS FINDA

Affected by Flags: Date Format (−42)

Remarks: If the level 1 argument is a real number *date*, FINDALARM returns the index of the first alarm due after 12:00 AM on that date. If the argument is a list { *date time* }, it returns the index of the first alarm due after that date and time. If the argument is the real number 0, FINDALARM returns the first *past-due* alarm.

For any of the three arguments, FINDALARM returns 0 if no alarm is found.

Related Commands: DELALARM, RCLALARM, STOALARM

FINISH

Finish Server Mode Command: Terminates Kermit Server mode in a device connected to an HP 48

Keyboard Access: ◀ I/O SERVR FINIS

Affected by Flags: I/O Device flag (−33), I/O Messages (−39)

Remarks: FINISH is used by a local Kermit device to tell a server Kermit (connected via the serial port or the IR port) to exit Server mode.

Related Commands: BAUD, CKSM, KGET, PARITY, PKT, RECN, RECV, SEND, SERVER

FIX

Fix Mode Command: Sets the number display format to Fix mode, which rounds the display to n decimal places.

{}

Level 1	→	Level 1
n	→	

Keyboard Access: ◀ MODES FMT FIX

Affected by Flags: None

Remarks: Fix mode shows n digits to the right of the fraction mark (decimal point), where $0 \leq n \leq 11$. (Values for n outside this range are rounded to the nearest integer.) A number is displayed or printed as follows:

(*sign*) *mantissa*

where the mantissa can be of any form. However, the calculator automatically displays a number in Scientific mode if either of the following is true:

- The number of digits to be displayed exceeds 12.
- A nonzero value rounded to n decimal places otherwise would be displayed as zero.

Example: The number 103.6 in Fix mode to four decimal places would appear as `103.6000`.

Related Commands: FIX, SCI, STD

FLOOR

Floor Function: Returns the greatest integer that is less than or equal to the argument.

{ }

Level 1	→	Level 1
x	→	*n*
x_unit	→	*n_unit*
'*symb*'	→	'FLOOR(*symb*)'

Keyboard Access: MTH REAL NXT NXT FLOOR

Affected by Flags: Numerical Results (−3)

Examples: `3.2 FLOOR` returns 3; `−3.2 FLOOR` returns −4.

Related Commands: CEIL, IP, RND, TRNC

FOR

FOR Definite Loop Structure Command: Starts FOR ... NEXT and FOR ... STEP definite loop structures.

	Level 2	Level 1	→	Level 1
FOR	x_{start}	x_{finish}	→	
NEXT			→	
FOR	x_{start}	x_{finish}	→	
STEP		$x_{increment}$	→	
STEP		'$symb_{increment}$'	→	

Keyboard Access:
To begin a definite loop:

PRG BRCH FOR FOR

To type FOR NEXT

PRG BRCH ↰ FOR

To type FOR STEP:

PRG BRCH ↱ FOR

Affected by Flags: None

Remarks: Definite loop structures execute a command or sequence of commands a specified number of times.

- A FOR ... NEXT loop executes a program segment a specified number of times using a local variable as the loop counter. You can use this variable within the loop. The syntax is this:

  x_{start} x_{finish} FOR *counter loop-clause* NEXT

  FOR takes x_{start} and x_{finish} from the stack as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Then, the loop clause is executed; *counter* can be referenced or have its value changed within the loop clause. NEXT increments *counter* by one, and then tests whether

counter is less than or equal to x_{finish}. If so, the loop clause is repeated (with the new value of *counter*).

When the loop is exited, *counter* is purged.

- FOR ... STEP works just like FOR ... NEXT, except that it lets you specify an increment value other than 1. The syntax is:

  x_{start} x_{finish} FOR *counter loop-clause* $x_{increment}$ STEP

  FOR takes x_{start} and x_{finish} from the stack as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Next, the loop clause is executed; *counter* can can be referenced or have its value changed within the loop clause. STEP takes $x_{increment}$ from the stack and increments *counter* by that value. If the argument of STEP is an algebraic expression or a name, it is automatically evaluated to a number.

  The increment value can be positive or negative. If the increment is positive, the loop is executed again when *counter* is less than or equal to x_{finish}. If the increment is negative, the loop is executed when *counter* is greater than or equal to x_{finish}.

  When the loop is exited, *counter* is purged.

Example: The following program sums all odd integers in the range 1 to 100:

```
« 0 1 100
 FOR I I + 2 STEP
»
```

Related Commands: NEXT, START, STEP

FP

Fractional Part Function: Returns the fractional part of the argument.

{}

Level 1	→	Level 1
x	→	*y*
x_unit	→	*y_unit*
'*symb*'	→	'FP(*symb*)'

Keyboard Access: MTH REAL NXT FP

Affected by Flags: Numerical Results (−3)

Remarks: The result has the same sign as the argument.

Examples: `-32.3 FP` returns `-.3`; `32.3_m FP` returns `.3_m`.

Related Commands: IP

FREE

Free RAM Card Command: Frees (makes *independent*) the previously merged RAM in port 1. FREE is provided for compatibility with the HP 48SX, which could merge RAM in port 2 as well. See FREE1.

Level 2	Level 1	→	Level 1
{ }	n_{port}	→	
{$name_{backup}$ ⋯ $n_{library}$ }	n_{port}	→	
$name_{backup}$	n_{port}	→	
$n_{library}$	n_{port}	→	

FREE

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Any prior contents of the port are moved into user memory. If level 2 specifies backup or library objects, those objects are moved from port 0 to the newly freed RAM port.

Related Commands: FREE1, MERGE1

FREE1

Free RAM Card Command: Frees (makes *independent*) the previously merged RAM in port 1. Any prior contents of the port are moved into user memory. If level 1 specifies backup or library objects, those objects are moved from port 0 to the newly freed RAM port.

Level 1	→	Level 1
$\{name_{backup} \cdots n_{library}\}$	→	
$name_{backup}$	→	
$n_{library}$	→	

Keyboard Access: ◀ LIBRARY FREE1

Affected by Flags: None

Remarks: The list in level 1 can be empty (in which case no objects are moved to the newly independent RAM) or it can contain any number of backup names and library numbers. Level 1 cannot be completely empty, however.

Related Commands: FREE, MERGE, MERGE1

FREEZE

Freeze Display Command: Freezes the part of the display specified by $n_{display\ area}$, so that it is not updated until a key is pressed.

{}

Level 1	→	Level 1
$n_{displayarea}$	→	

Keyboard Access: PRG NXT OUT FREEZ

Affected by Flags: None

Remarks: Normally, the stack display is updated as soon as the calculator is ready for data input. For example, when HALT stops a running program, or when a program ends, any displayed messages are cleared. The FREEZE command "freezes" a part or all of the display so that it is not updated until a key is pressed. This allows, for example, a prompting message to persist after a program halts to await data input.

$n_{display\ area}$ is the sum of the value codes for the areas to be frozen:

Display Area	Value Code
Status area	1
Stack/Command-line area	2
Menu area	4

So, for example, 2 FREEZE freezes the stack/command-line area, 3 FREEZE freezes the status area and the stack/command-line area, and 7 FREEZE freezes all three areas.

Values of $n_{display\ area} \geq 7$ or ≤ 0 freeze the entire display (are equivalent to value 7). To freeze the graphics display, you must freeze the status and stack/command-line areas (by entering 3), or the entire display (by entering 7).

Examples:

This program:

```
« "Ready for data" 1 DISP 1 FREEZE HALT »
```

displays the contents of the string in the top line of the display, then freezes the status area so that the string contents persist in the display after HALT is executed:

This program:

```
« 4 # 0d # 0d 3 PVIEW 7 FREEZE »
```

selects the graphics display and then freezes the entire display so that the graphics display persists after the program ends. (If FREEZE was not executed, the stack display would be selected after the program ends.)

To use FREEZE with PVIEW (or any graphics display), you must enter 3 or 7.

Related Commands: CLLCD, DISP, HALT

FS?

Flag Set? Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear.

{}

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	0/1

Keyboard Access:

PRG TEST NXT NXT FS?

MODES FLAG FS?

Affected by Flags: None

Related Commands: CF, FC?, FC?C, FS?C, SF

FS?C

Flag Set? Clear Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear. After testing, clears the flag.

{}

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	0/1

Keyboard Access:

PRG TEST NXT NXT FS?C

MODES FLAG FS?C

Affected by Flags: None

Example: If flag −44 is set, −44 FS?C returns 1 to level 1 and clears flag −44.

Related Commands: CF, FC?, FC?C, FS?, SF

FUNCTION

Function Plot Type Command: Sets the plot type to FUNCTION.

Keyboard Access: PLOT PTYPE FUNC

Affected by Flags: Simultaneous Plotting (−28), Curve Filling (−31)

Remarks: When the plot type is FUNCTION, the DRAW command plots the current equation as a real-valued function of one real variable. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

FUNCTION

For plot type FUNCTION, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (-6.5,-3.1).
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of *indep* is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command FUNCTION places the name FUNCTION in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The current equation is plotted as a function of the variable specified in *indep*. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the values in (x_{min}, y_{min}) and (x_{max}, y_{max}) (the display range) are used. Lines are drawn between plotted points unless flag −31 is set.

If *EQ* contains an expression or program, the expression or program is evaluated in Numerical Results mode for each value of the independent variable to give the values of the dependent variable. If *EQ* contains an equation, the plotting action depends on the form of the equation, as shown in the following table.

Form of Current Equation	Plotting Action
'*expr=expr*'	Each expression is plotted separately. The intersection of the two graphs shows where the expressions are equal.
'*name=expr*'	Only the expression is plotted.
'*indep=constant*'	A vertical line is plotted.

If flag −28 is set, all equations are plotted simultaneously.

If the independent variable in the current equation represents a unit object, you must specify the units by storing a unit object in the corresponding variable in the current directory. For example, if the current equation is '*X*+3_m', and you want *X* to represent some number of inches, you would store 1_in (the number part of the unit object is ignored) in *X*. For each plotted point, the numerical value of the independent variable is combined with the specified unit (inches in this example) before the current equation is evaluated. If the result is a unit object, only the number part is plotted.

Related Commands: BAR, CONIC, DIFFEQ, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

GET

Get Element Command: Returns from the level 2 array or list (or named array or list) the real or complex number z_{get} or object obj_{get} whose position is specified in level 1.

Level 2	Level 1	→	Level 1
[[*matrix*]]	$n_{position}$	→	z_{get}
[[*matrix*]]	{ n_{row} m_{col} }	→	z_{get}
'$name_{matrix}$'	$n_{position}$	→	z_{get}
'$name_{matrix}$'	{ n_{row} m_{col} }	→	z_{get}
[*vector*]	$n_{position}$	→	z_{get}
[*vector*]	{ $n_{position}$ }	→	z_{get}
'$name_{vector}$'	$n_{position}$	→	z_{get}
'$name_{vector}$'	{ $n_{position}$ }	→	z_{get}
{ *list* }	$n_{position}$	→	obj_{get}
{ *list* }	{ $n_{position}$ }	→	obj_{get}
'$name_{list}$'	$n_{position}$	→	obj_{get}
'$name_{list}$'	{ $n_{position}$ }	→	obj_{get}

Keyboard Access: PRG LIST ELEM GET

Affected by Flags: None

Remarks: For matrices, $n_{position}$ is incremented in *row* order.

Examples: [[2 3 7][3 2 9][2 1 3]] { 2 3 } GET returns 9.

[[2 3 7][3 2 9][2 1 3]] 8 GET returns 1.

{ A B C D E } { 1 } GET returns 'A'

Related Commands: GETI, PUT, PUTI

GETI

Get and Increment Index Command: Returns from the level 2 array or list (or named array or list) the real or complex number z_{get} or object obj_{get} whose position is specified in level 1, along with the level 2 argument and the next position in that argument.

Level 2	Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]]	n_{pos1}	→	[[*matrix*]]	n_{pos2}	z_{get}
[[*matrix*]]	{ n_r m_c }$_1$	→	[[*matrix*]]	{ n_r m_c }$_2$	z_{get}
'$name_{mtrx}$'	n_{pos1}	→	'$name_{mtrx}$'	n_{pos2}	z_{get}
'$name_{mtrx}$'	{ n_r m_c }$_1$	→	'$name_{mtrx}$'	{ n_r m_c }$_2$	z_{get}
[*vect*]	n_{pos1}	→	[*vect*]	n_{pos2}	z_{get}
[*vect*]	{ n_{pos1} }	→	[*vect*]	{ n_{pos2} }	z_{get}
'$name_{vect}$'	n_{pos1}	→	'$name_{vect}$'	n_{pos2}	z_{get}
'$name_{vect}$'	{ n_{pos1} }	→	'$name_{vect}$'	{ n_{pos2} }	z_{get}
{ *list* }	n_{pos1}	→	{ *list* }	n_{pos2}	obj_{get}
{ *list* }	{ n_{pos1} }	→	{ *list* }	{ n_{pos2} }	obj_{get}
'$name_{list}$'	n_{pos1}	→	'$name_{list}$'	n_{pos2}	obj_{get}
'$name_{list}$'	{ n_{pos1} }	→	'$name_{list}$'	{ n_{pos2} }	obj_{get}

Keyboard Access: PRG LIST ELEM GETI

Affected by Flags: Index Wrap Indicator (−64)

The Index Wrap Indicator flag is cleared on each execution of GETI *until* the position index wraps to the first position in the argument, at which point the flag is set. The next execution of GETI clears the flag again.

Remarks: For matrices, the position is incremented in *row* order.

Related Commands: GET, PUT, PUTI

GOR

Graphics OR Command: Superimposes $grob_1$ onto $grob_{target}$ or *PICT*, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{target}$ or *PICT*.

Level 3	Level 2	Level 1	→	Level 1
$grob_{target}$	{ #n #m }	$grob_1$	→	$grob_{result}$
$grob_{target}$	(x,y)	$grob_1$	→	$grob_{result}$
PICT	{ #n #m }	$grob_1$	→	
PICT	(x,y)	$grob_1$	→	

Keyboard Access: PRG GROB GOR

Affected by Flags: None

Remarks: GOR uses a logical OR to determine the state (on or off) of each pixel in the overlapping portion of the argument graphics object.

If the level 3 argument is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the level 3 argument is *PICT*, no result is returned to the stack.

Any portion of $grob_1$ that extends past $grob_{target}$ or *PICT* is truncated.

Related Commands: GXOR, REPL, SUB

GRAD

Grads Mode Command: Sets Grads angle mode.

Keyboard Access: ← MODES ANGL GRAD

Affected by Flags: None

Remarks: GRAD clears flag −17 and sets flag −18, and displays the GRAD annunciator.

In Grads angle mode, real-number arguments that represent angles are interpreted as grads, and real-number results that represent angles are expressed in grads.

Related Commands: DEG, RAD

GRAPH

Picture Environment Command: Selects the Picture environment (selects the graphics display and activates the graphics cursor and Picture menu).

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: GRAPH is provided for compatibility with the HP 48S series, and is the same as the PICTURE command. See PICTURE.

Related Commands: PICTURE, PVIEW, TEXT

GRIDMAP

GRIDMAP Plot Type Command: Sets plot type to GRIDMAP.

Keyboard Access: ◀ PLOT NXT 3D TYPE GRID

Affected by Flags: None

Remarks: When plot type is set GRIDMAP, the DRAW command plots a mapping grid representation of a 2-vector-valued function of two variables. GRIDMAP requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

$\{ x_{left}\ x_{right}\ y_{near}\ y_{far}\ z_{low}\ z_{high}\ x_{min}\ x_{max}\ y_{min}\ y_{max}\ x_{eye}\ y_{eye}\ z_{eye}\ x_{step}\ y_{step} \}$.

For plot type GRIDMAP, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are real numbers the specify the input region's width. The default value is $\{-1, 1\}$.
- y_{min} and y_{max} are real numbers that specify the input region's depth. The default value is $\{-1, 1\}$.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which you view the graph.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted. These can be used instead of (or in combination with) RES.

The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

$\{ \{x_{min}, y_{min}\} \{x_{max}, y_{max}\}$ *indep res axes ptype depend* $\}$

For plot type GRIDMAP, the elements of *PPAR* are used as follows:

- $\{x_{min}, y_{min}\}$ is not used.

- $\{x_{max}, y_{max}\}$ is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command GRIDMAP places the command name GRIDMAP in *PPAR*.
- *depend* is a name specifying the dependent variable. The default value is *Y*

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

→GROB

Stack to Graphics Object Command: Creates a graphics object representing the level 2 object, where the argument $n_{char\ size}$ specifies the character size of the representation.

{}

Level 2	Level 1	→	Level 1
obj	$n_{charsize}$	→	*grob*

Keyboard Access: PRG TYPE →GRO

Affected by Flags: None

Remarks: $n_{char\ size}$ can be 0, 1 (small), 2 (medium), or 3 (large). $n_{char\ size} = 0$ is the same as $n_{char\ size} = 3$, except for unit objects

and algebraic objects, where 0 specifies the EquationWriter application picture.

Example: This program:

```
« 'Y=3*X^2' 0 →GROB PICT STO { } PVIEW »
```

returns a graphics object to the stack representing the EquationWriter application picture of 'Y=3*X^2', then stores the graphics object in *PICT* and shows it in the graphics display with scrolling activated.

Related Commands: →LCD, LCD→

GXOR

Graphics Exclusive OR Command: Superimposes $grob_1$ onto $grob_{target}$ or *PICT*, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{target}$ or *PICT*.

Level 3	Level 2	Level 1	→	Level 1
$grob_{target}$	{ #n #m }	$grob_1$	→	$grob_{result}$
$grob_{target}$	(x,y)	$grob_1$	→	$grob_{result}$
PICT	{ #n #m }	$grob_1$	→	
PICT	(x,y)	$grob_1$	→	

Keyboard Access: PRG GRAPH GXOR

Affected by Flags: None

Remarks: GXOR is used for creating cursors, for example, to make the cursor image appear dark on a light background and light on a dark background. Executing GXOR again with the same image restores the original picture.

GXOR uses a logical exclusive OR to determine the state of the pixels (on or off) in the overlapping portion of the argument graphics objects.

Any portion of $grob_1$ that extends past $grob_{target}$ or *PICT* is truncated.

If the level 3 argument (the target graphics object) is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the level 3 argument is *PICT*, no result is returned to the stack.

Example: This program:

```
« ERASE PICT NEG PICT { # 0d #0d }
GROB 5 × 5 11A040A011 GXOR LASTARG GXOR »
```

turns on (makes dark) every pixel in *PICT*, then superimposes a 5 × 5 graphics object on *PICT* at pixel coordinates { # 0d # 0d }. Each on-pixel in the 5 by 5 graphics object turns off (makes light) the corresponding pixel in *PICT*. Then, the original picture is restored by executing GXOR again with the same arguments.

Related Commands: GOR, REPL, SUB

*H

Multiply Height Command: Multiplies the vertical plot scale by x_{factor}.

{}

Level 1	→	Level 1
x_{factor}	→	

Keyboard Access: ◀ PLOT PTYPE NXT *H

Affected by Flags: None

Remarks: Executing *H changes the *y*-axis display range—the y_{min} and y_{max} components of the first two complex numbers in the reserved variable *PPAR*. The plot origin (the user-unit coordinate of the center pixel) is not changed.

Related Commands: AUTO, *W, YRNG

HALT

Halt Program Command: Halts program execution.

Keyboard Access: PRG NXT RUN HALT

Affected by Flags: None

Remarks: Program execution is halted at the location of the HALT command in the program. The HALT annunciator is turned on. Program execution is resumed by executing CONT (usually by pressing ◀ CONT). Executing KILL (usually by pressing PRG NXT RUN KILL) cancels all halted programs.

Related Commands: CONT, KILL

HEAD

First Listed Element Command: Returns the first element of a list or string.

Level 1	→	Level 1
{ obj_1 ... obj_n }	→	obj_1
"*string*"	→	"$element_1$"

Keyboard Access:

PRG LIST ELEM NXT HEAD

◀ CHARS NXT HEAD

Affected by Flags: None

Example: "Dead" HEAD returns "D"

The following program takes a list of coordinates { A B C } that define a right triangle, and finds the length of the hypotenuse AC.

```
« DUP HEAD SWAP REVLIST HEAD - ABS »
```

For example, entering { (0,0) (0,3) (3,4) } returns 5.

Related Commands: TAIL

HEX

Hexadecimal Mode Command: Selects hexadecimal base for binary integer operations. (The default base is decimal.)

Keyboard Access: MTH BASE HEX

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: Binary integers require the prefix #. Binary integers entered and returned in hexadecimal base automatically show the suffix h. If the current base is not hexadecimal, then you can enter a hexadecimal number by ending it with h. It will be displayed in the current base when it is entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Related Commands: BIN, DEC, OCT, RCWS, STWS

HISTOGRAM

Histogram Plot Type Command: Sets the plot type to HISTOGRAM.

Keyboard Access: ◀ PLOT NXT STAT PTYPE HISTO

Affected by Flags: None

Remarks: When the plot type is HISTOGRAM, the DRAW command creates a histogram using data from one column of the current statistics matrix (reserved variable ΣDAT). The column is specified by the first parameter in the reserved variable ΣPAR (using the XCOL command). The plotting parameters are specified in the reserved variable $PPAR$, which has the form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

HISTOGRAM

For plot type HISTOGRAM, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (-6.5,-3.1).
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers that specify the minimum and maximum values of the data to be plotted. The default value of *indep* is *X*.
- *res* is a real number specifying the bin size, in user-unit coordinates, or a binary integer specifying the bin size in pixels. The default value is 0, which specifies the bin size to be 1/13 of the difference between the specified minimum and maximum values of the data.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin; a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command HISTOGRAM places the command name HISTOGRAM in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The frequency of the data is plotted as bars, where each bar represents a collection of data points. The base of each bar spans the values of the data points, and the height indicates the number of data points. The width of each bar is specified by *res*. The overall maximum and minimum values for the data can be specified by *indep*; otherwise, the values in (x_{min}, y_{min}) and (x_{max}, y_{max}) are used.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

HISTPLOT

Draw Histogram Plot Command: Plots a frequency histogram of the specified column in the current statistics matrix (reserved variable *ΣDAT*).

Keyboard Access: ← STAT PLOT HISTP

Affected by Flags: None

Remarks: The data column to be plotted is specified by XCOL and is stored as the first parameter in the reserved variable *ΣPAR*. If no data column is specified, column 1 is selected by default. The *y*-axis is autoscaled and the plot type is set to HISTOGRAM.

HISTPLOT plots *relative* frequencies, using 13 bins as the default number of partitions. The RES command lets you specify a different number of bins by specifying the bin width. To plot a frequency histogram with *numerical* frequencies, store the frequencies in *ΣDAT* and execute BINS and then BARPLOT.

When HISTPLOT is executed from a program, the graphics display, which shows the resultant plot, does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

Related Commands: BARPLOT, BINS, FREEZE, PICTURE, PVIEW, RES, SCATRPLOT, XCOL

HMS+

Hours-Minutes-Seconds Plus Command: Returns the sum of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

{}

Level 2	Level 1	→	Level 1
HMS_1	HMS_2	→	$HMS_1 + HMS_2$

HMS+

Keyboard Access: ◀ TIME NXT HMS+

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, →HMS, HMS−

HMS−

Hours-Minutes-Seconds Minus Command: Returns the difference of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

{ }

Level 2	Level 1	→	Level 1
HMS_1	HMS_2	→	$HMS_1 - HMS_2$

Keyboard Access: ◀ TIME NXT HMS−

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, →HMS, HMS+

HMS→

Hours-Minutes-Seconds to Decimal Command: Converts a real number in hours-minutes-seconds format to its decimal form (hours or degrees with a decimal fraction).

{ }

Level 1	→	Level 1
HMS	→	*x*

Keyboard Access: ◀ TIME NXT HMS+

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: →HMS, HMS+, HMS−

→HMS

Decimal to Hours-Minutes-Seconds Command: Converts a real number representing hours or degrees with a decimal fraction to hours-minutes-seconds format.

{}

Level 1	→	Level 1
x	→	*HMS*

Keyboard Access: ◀ TIME NXT HMS▸

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, HMS+, HMS−

HOME

HOME Directory Command: Makes the *HOME* directory the current directory.

Keyboard Access: ▶ HOME

Affected by Flags: None

Related Commands: CRDIR, PATH, PGDIR, UPDIR

i

i Function: Returns the symbolic constant *i* or its numerical representation, (0, 1).

Level 1	→	Level 1
	→	'i'
	→	(0,1)

Keyboard Access:

α ◀ I

MTH NXT CONS I

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

Evaluating *i* returns its numerical representation if flag −2 or −3 is set; otherwise, its symbolic representation is returned.

Related Commands: e, MAXR, MINR, π

IDN

Identity Matrix Command: Returns an identity matrix; that is, a square matrix with its diagonal elements equal to 1 and its off-diagonal elements equal to 0.

{}

Level 1	→	Level 1
n	→	[[*R-matrix*$_{identity}$]]
[[*matrix*]]	→	[[*matrix*$_{identity}$]]
'*name*'	→	

IDN

Keyboard Access: (MTH) MATR MAKE IDN

Affected by Flags: None

Remarks: The result is either a new square matrix, or it's an existing square matrix with its elements replaced by the elements of the identity matrix, according to the argument in level 1.

- **Creating a new matrix:** If the argument is a real number n, a new real identity matrix is returned to level 1, with its number of rows and number of columns equal to n.
- **Replacing the elements of an existing matrix:** If the argument is a square matrix, an identity matrix of the same dimensions is returned. If the original matrix is complex, the resulting identity matrix will also be complex, with diagonal values (1,0).

  If the argument is a name, the name must identify a variable containing a square matrix. In this case, the elements of the matrix are replaced by those of the identity matrix (complex if the original matrix is complex).

Related Commands: CON

IF

IF Conditional Structure Command: Starts IF ... THEN ... END and IF ... THEN ... ELSE ... END conditional structures.

	Level 1	→	Level 1
IF		→	
THEN	*T/F*	→	
END		→	
		→	
IF		→	
THEN	*T/F*	→	
ELSE		→	
END		→	

Keyboard Access: (PRG) BRCH IF IF

Affected by Flags: None

Remarks: *Conditional structures*, used in combination with program tests, enable a program to make decisions.

- IF ... THEN ... END executes a sequence of commands only if a test returns a nonzero (true) result. The syntax is:

  IF *test-clause* THEN *true-clause* END

  IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is nonzero, the true clause is executed. Otherwise, program execution resumes following END.

- IF ... THEN ... ELSE ... END executes one sequence of commands if a test returns a true (nonzero) result, or another sequence of commands if that test returns a false (zero) result. The syntax is:

  IF *test-clause* THEN *true-clause* ELSE *false-clause* END

  IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is nonzero, the true clause is executed. Otherwise, the false clause is executed. After the appropriate clause is executed, execution resumes following END.

The test clause can be a command sequence (for example, A B ≤) or an algebraic (for example, 'A≤B'). If the test clause is an algebraic, it is *automatically evaluated* to a number (→NUM or EVAL isn't necessary).

Related Commands: CASE, ELSE, END, IFERR, THEN

IFERR

If Error Conditional Structure Command Starts IFERR ... THEN ... END and IFERR ... THEN ... ELSE ... END error trapping structures.

Keyboard Access: PRG NXT ERROR IFERR

Affected by Flags: Last Arguments (−55)

Remarks: *Error trapping* structures enable program execution to continue after a "trapped" error occurs.

- IFERR ... THEN ... END executes a sequence of commands if an error occurs. The syntax of IFERR ... THEN ... END is:

  IFERR *trap-clause* THEN *error-clause* END

  If an error occurs during execution of the trap clause:

  1. The error is ignored.
  2. The remainder of the trap clause is discarded.
  3. The key buffer is cleared.
  4. If any or all of the display is "frozen" (by FREEZE), that state is canceled.
  5. If Last Arguments is enabled, the arguments to the command that caused the error are returned to the stack.
  6. Program execution jumps to the error clause.

  The commands in the error clause are executed only if an error is generated during execution of the trap clause.

- IFERR ... THEN ... ELSE ... END executes one sequence of commands if an error occurs or another sequence of commands if an error does not occur. The syntax of IFERR ... THEN ... ELSE ... END is:

  IFERR *trap-clause* THEN *error-clause* ELSE *normal-clause* END

  If an error occurs during execution of the trap clause, the same six events listed above occur.

  If no error occurs, execution jumps to the normal clause at the completion of the trap clause.

Example: The following program uses IFERR much like the built-in linear system of equations solver. The program takes a result vector and a matrix of coefficients and returns a least-squares solution to the equations.

```
« → a b
  « IFERR a b / THEN LSQ END
  »
»
```

Related Commands: CASE, ELSE, END, IF, THEN

IFFT

Inverse Discrete Fourier Transform Command: Computes the one- or two-dimensional inverse discrete Fourier transform of an array.

{}

Level 1	→	Level 1
[*array*]$_1$	→	[*array*]$_2$

Keyboard Access: MTH NXT FFT IFFT

Affected by Flags: None

Remarks: If the argument is an N-vector or an $N\times1$ or $1\times N$ matrix, IFFT computes the one-dimensional inverse transform. If the argument is an $M\times N$ matrix, IFFT computes the two-dimensional inverse transform. M and N must be integral powers of 2.

The one-dimensional inverse discrete Fourier transform of an N-vector Y is the N-vector X where:

$$X_n = \frac{1}{N}\sum_{k=0}^{N-1} Y_k e^{\frac{2\pi i k n}{N}} \qquad i = \sqrt{-1}$$

for $n = 0,\ 1,\ \ldots,\ N-1$.

IFFT

The two-dimensional inverse discrete Fourier transform of an $M \times N$ matrix Y is the $M \times N$ matrix X where:

$$X_{mn} = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} Y_{kl} e^{\frac{2\pi i k m}{M}} e^{\frac{2\pi i l n}{N}} \qquad i = \sqrt{-1}$$

for $m = 0,\ 1,\ \ldots,\ M-1$ and $n = 0,\ 1,\ \ldots,\ N-1$.

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The IFFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

Related Commands: FFT

IFT

IF-THEN Command: Executes *obj* if *T/F* is nonzero. Discards *obj* if *T/F* is zero.

Level 2	Level 1	→	Level 1
T/F	*obj*	→	*It depends!*

Keyboard Access: PRG BRCH NXT IFT

Affected by Flags: None

Remarks: IFT lets you execute in stack syntax the decision-making process of the IF ... THEN ... END conditional structure. The "true clause" is *obj* in level 1.

Example: `« X 0 > "Positive" IFT »` puts `"Positive"` in level 1 if X contains a positive real number.

Related Commands: IFTE

IFTE

IF-THEN-ELSE Function: Executes the *obj* on level 2 if *T/F* is nonzero. Executes the *obj* on level 1 if *T/F* is zero.

Level 3	Level 2	Level 1	→	Level 1
T/F	obj_{true}	obj_{false}	→	*It depends!*

Keyboard Access: PRG BRCH NXT IFTE

Affected by Flags: None

Remarks: IFTE lets you execute in stack syntax the decision-making process of the IF ... THEN ... ELSE ... END conditional structure. The "true clause" is obj_{true} in level 2. The "false clause" is obj_{false} in level 1.

IFTE is also allowed in algebraic expressions, with the following syntax:

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

When an algebraic containing IFTE is evaluated, its first argument *test* is evaluated to a test result. If it returns a nonzero real number, *true-clause* is evaluated. If it returns zero, *false-clause* is evaluated.

Examples: The command sequence `X 0 ≥ "Positive" "Negative" IFTE` leaves `"Positive"` on the stack if X contains a non-negative real number, or `"Negative"` if X contains a negative real number.

The algebraic `'IFTE(X≠0,SIN(X)/X,1)'` returns the value of $\sin(x)/x$, even for $x = 0$, which would normally cause an `Infinite Result` error.

Related Commands: IFT

IM

Imaginary Part Function: Returns the imaginary part of its (complex) argument.

{}

Level 1	→	Level 1
x	→	0
(*x*, *y*)	→	*y*
[*R-array*]	→	[*R-array*]
[*C-array*]	→	[*R-array*]
'*symb*'	→	'IM(*symb*)'

Keyboard Access: MTH NXT CMPL IM

Affected by Flags: Numerical Results (−3)

Remarks: If the argument is an array, IM returns a real array, the elements of which are equal to the imaginary parts of the corresponding elements of the argument array. If the argument array is real, all of the elements of the result array are zero.

Related Commands: C→R, RE, R→C

INCR

Increment Command: Takes a variable on level 1, adds 1, stores the new value back into the original variable, and returns the new value to level 1.

{}

Level 1	→	Level 1
'*name*'	→	$x_{increment}$

Keyboard Access: ↶ MEMORY MATH INCR

Affected by Flags: None

Remarks: The value in *name* must be a real number.

Example: If 35.7 is stored in *A*, 'A' INCR returns 36.7.

Related Commands: DECR

INDEP

Independent Variable Command: Specifies the independent variable and its plotting range.

Level 2	Level 1	→	Level 1
	'*global*'	→	
	{ *global* }	→	
	{ *global* x_{start} x_{end} }	→	
	{ x_{start} x_{end} }	→	
x_{start}	x_{end}	→	

Keyboard Access: ↶ PLOT PPAR INDEP

Affected by Flags: None

Remarks: The specification for the independent variable name and its plotting range is stored as the third parameter in the reserved variable *PPAR*. If the argument to INDEP is a:

- Global variable name, that name replaces the independent variable entry in *PPAR*.
- List containing a global name, that name replaces the independent variable name but leaves unchanged any existing plotting range.
- List containing a global name and two real numbers, that list replaces the independent variable entry.

INDEP

- List containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the independent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is *X*.

Related Commands: DEPND

INFORM

User-Defined Dialog Box Command: Creates a user-defined input form (dialog box).

Lvl 5	Lvl 4	Lvl 3	Lvl 2	Lvl 1 →	Lvl 2	Lvl 1
'title'	{ s_1 s_2...s_n }	format	{ resets }	{ init } →	{ vals }	1
'title'	{ s_1 s_2...s_n }	format	{ resets }	{ init } →		0

Keyboard Access: PRG NXT [illegible] INFORM

Affected by Flags: None

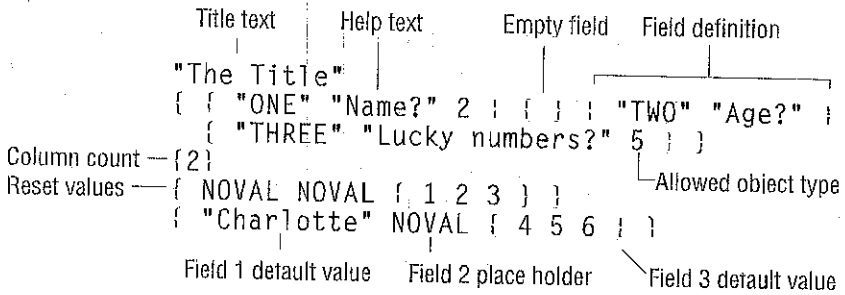
Remarks: INFORM creates a standard dialog box based upon the following specifications:

Variable	Function
"title"	Title. This appears at the top of the dialog box.
{s_1 s_2 ... s_n}	Field definitions. A field definition (s_x) can have two formats: *"label"*, a field label, or { *"label"* *"helpInfo"* $type_0$ $type_1$ ... $type_n$ }, a field label with optional help text that appears near the bottom of the screen, and an optional list of valid object types for that field. If object types aren't specified, all object types are valid. For infomation about object types, see the TYPE command.
	When creating a multi-column dialog box, you can span columns by using an empty list as a field definition. A field that appears to the left of an empty field automatically expands to fill the empty space.
format	Field format information. This is the number *col* or a list of the form { *col tabs* }: *col* is the number of columns the dialog box has, and *tabs* optionally specifies the number of tab stops between the labels and the highlighted fields. This list can be empty. *col* defaults to 1 and *tabs* defaults to 3.
{ *resets* }	Default values displayed when RESET is selected. Specify reset values in the list in the same order as the fields were specified. To specify no value, use the NOVAL command as a place holder. This list can be empty.
{ *init* }	Initial values displayed when the dialog box appears. Specify initial values in the list in the same order as the fields were specified. To specify no value, use the NOVAL command as a place holder. This list can be empty.

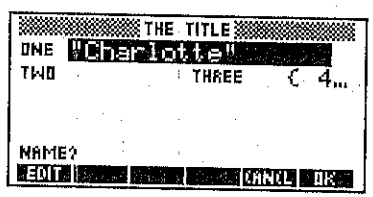
If you exit the dialog box by selecting OK or ENTER, INFORM returns the field values { *vals* } on level 2, and puts a 1 on level 1. (If a field is empty, NOVAL is returned as a place holder.) If you exit the dialog box by selecting CANCEL or CANCEL, INFORM returns 0.

INFORM

Example: If the following five lines are on the stack:



Pressing INFORM would produce:



Related Commands: CHOOSE, INPUT, NOVAL, TYPE

INPUT

Input Command: Prompts for data input to the command line and prevents the user access to stack operations.

Level 2	Level 1	→	Level 1
"stack prompt"	*"command-line prompt"*	→	*"result"*
"stack prompt"	{ *list*$_{command-line}$ }	→	*"result"*

Keyboard Access: PRG NXT IN INPUT

Affected by Flags: None

Remarks: When INPUT is executed, the stack area is blanked and program execution is suspended for data input to the command line. The contents of *"stack prompt"* are displayed at the top of the stack area. Depending on the level 1 argument, the command line may also contain the contents of a string, or it may be empty. Pressing ENTER resumes program execution and returns the contents of the command line in string form to level 1.

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

- *"command-line prompt"*, whose contents are placed in the command line for prompting when the program pauses.
- Either a *real number*, or a *list containing two real numbers*, that specifies the initial cursor position in the command line:
  - A real number *n* at the *n*th character from the left end of the first row (line) of the command line. A *positive n* specifies the insert cursor; a *negative n* specifies the replace cursor. 0 specifies the end of the command-line string.
  - A list that specifies the initial row and column position of the cursor: the first number in the list specifies a row in the command line (1 specifies the first row of the command line); the second number counts by characters from the left end of the specified line. 0 specifies the end of the command-line string in the specified row. A positive row number specifies the insert cursor; a negative row number specifies the replace cursor.
- One or more of the parameters ALG, α, or V, entered as unquoted names:
  - ALG activates Algebraic/Program-entry mode.
  - α (α → A) specifies alpha lock.
  - V verifies if the characters in the result string "result", without the " delimiters, compose a valid object or objects. If the result-string characters do not compose a valid object or objects, INPUT displays the Invalid Syntax warning and prompts again for data.

INPUT

You can choose to specify as few as one of the level-1 list parameters. The default states for these parameters are:

- Blank command line.
- Insert cursor placed at the end of the command-line prompt string.
- Program-entry mode.
- Result string not checked for invalid syntax.

If you specify *only* a command-line prompt string for the level 1 argument, you don't need to put it in a list.

Related Commands: PROMPT, STR→

INV

Inverse (1/x) Analytic Function: Returns the reciprocal or the matrix inverse.

{}

Level 1	→	Level 1
z	→	*1/z*
[[*matrix*]]	→	[[*matrix*]] $^{-1}$
'symb'	→	'INV(*symb*)'
x_unit	→	*1/x_1/unit*

Keyboard Access: (1/x)

Affected by Flags: Numerical Results (−3)

Remarks: For a *complex* argument (x, y), the inverse is the complex number $\left(\frac{x}{x^2+y^2}, \frac{-y}{x^2+y^2}\right)$.

Matrix arguments must be square (real or complex). The computed inverse matrix A^{-1} satisfies $A \times A^{-1} = I_n$, where I_n is the $n \times n$ identity matrix.

Related Commands: SINV, /

IP

Integer Part Function: Returns the integer part of its argument.

{}

Level 1	→	Level 1
x	→	*n*
x_unit	→	*n_unit*
'symb'	→	'IP(*symb*)'

Keyboard Access: (MTH) REAL (NXT) IP

Affected by Flags: Numerical Results (−3)

Remarks: The result has the same sign as the argument.

Example: `32.3_m IP` returns `32_m`.

Related Commands: FP

IR

Infrared/Serial Transmission Command: Directs I/O and printer output to the infrared or serial port ("wire").

Keyboard Access: (←)(I/O) I/O FM IR

Affected by Flags: I/O Device (−33), Printing Device (−34)

Remarks: Toggles between IR and wire.

For more information, refer also to the reserved variable *IOPAR* (*I/O parameters*) in appendix D, "Reserved Variables."

Related Commands: BAUD, CKSM, PARITY, TRANSIO

ISOL

Isolate Variable Command: Returns an algebraic '$symb_2$' that rearranges '$symb_1$' to "isolate" the first occurrence of variable *global*.

{}

Level 2	Level 1	→	Level 1
'$symb_1$'	'*global*'	→	'$symb_2$'

Keyboard Access: ↰ SYMBOLIC ISOL

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

When flag −3 is set, symbolic results are evaluated to real numbers. This means that the = sign is evaluated. If *global* or any other variable in the result equation is formal, an Undefined Name error results; if *global* and all other variables have values, a numerical result is returned from the calculation *global* − *expression*. This result has limited value. In general, execute ISOL with flag −3 clear.

Remarks: The result '$symb_2$' is an equation of the form '*global=expression*'. If *global* appears more than once, then '$symb_2$' is effectively the right side of an equation obtained by rearranging and solving '$symb_1$' to isolate the first occurrence of *global* on the left side of the equation.

If '$symb_1$' is an expression, it is treated as the left side of an equation '$symb_1$=0'.

If *global* appears in the argument of a function within '$symb_1$', that function must be an *analytic* function—a function for which the HP 48 provides an inverse. Thus ISOL cannot solve 'IP(X)=0' for X, since IP has no inverse.

Related Commands: COLCT, EXPAN, QUAD, SHOW

KERRM

Kermit Error Message Command: Returns the text of the most recent Kermit error packet.

Level 1	→	Level 1
	→	"*error-message*"

Keyboard Access: ↰ I/O NXT KERR

Affected by Flags: None

Remarks: If a Kermit transfer fails due to an error packet sent from the connected Kermit device to the HP 48, then executing KERRM retrieves and displays the error message. (Kermit errors not in packets are retrieved by ERRM rather than KERRM.)

Related Commands: FINISH, KGET, PKT, RECN, RECV, SEND, SERVER

KEY

Key Command: Returns to level 1 a test result and, if a key is pressed, returns to level 2 the row-column location x_{nm} of that key.

Level 1	→	Level 2	Level 1
	→	x_{nm}	1
	→		0

Keyboard Access: PRG NXT TIME KEY

Affected by Flags: None

Remarks: KEY returns a false result (0) to level 1 until a key is pressed. When a key is pressed, it returns a true result (1) to level

KEY

1 and x_{nm} to level 2. The result x_{nm} is a two-digit number that identifies the row and column location of the key just pressed. Unlike WAIT, which returns a three-digit number that identifies alpha and shifted keyboard planes, KEY returns the row-column location of *any* key pressed, including ◀, ▶, and α.

Example: The program « DO UNTIL KEY END 71 SAME » returns 1 to the stack if the ◀ key is pressed while the indefinite loop is running.

Related Commands: WAIT

KGET

Kermit Get Command: Used by a local Kermit to get a Kermit server to transmit the named object(s).

Level 1	→	Level 1
'name'	→	
"name"	→	
{ $name_{old}$ $name_{new}$ }	→	
{ $name_1$ ... $name_n$ }	→	
{{ $name_{old}$ $name_{new}$ } name ... }	→	

Keyboard Access: ◀ I/O SERVR KGET

Affected by Flags: I/O Device (−33), RECV Overwrite (−36), I/O Messages (−39)

I/O Data Format (−35) also affects KGET as follows:

- If the server is an HP 48, then the server's flag −35 affects KGET.
- If the server is not an HP 48 but the file being transferred originated from an HP 48, flag −35 has no effect.
- If the server is not an HP 48 and the file being transferred does not have the *%%HP...;* header, flag −35 tells the HP 48 whether to attempt parsing the incoming data.

Remarks: To rename an object when the local device gets it, include the old and new names in an embedded list. For example, « { AAA BBB } KGET gets the variable named *AAA* but changes its name to *BBB*. « { AAA BBB } { CCC } KGET gets *AAA* as *BBB* and gets *CCC* under its own name. (If the original name is not legal on the HP 48, enter it as a string.)

Related Commands: BAUD, CKSM, FINISH, PARITY, RECN, RECV, SEND, SERVER, TRANSIO

KILL

Cancel Halted Programs Command: Cancels all currently halted programs. (Halted programs are typically canceled by pressing PRG NXT RUN KILL.) If KILL is executed within a program, that program is also canceled.

Keyboard Access: PRG NXT RUN KILL

Affected by Flags: None

Remarks: Canceled programs cannot be resumed.

KILL cancels *only* halted programs and the program from which KILL was executed, if any. Commands that halt programs are HALT and PROMPT.

Suspended programs cannot be canceled. Commands that suspend programs are INPUT and WAIT.

Related Commands: CONT, DOERR, HALT, PROMPT

LABEL

Label Axes Command: Labels axes in *PICT* with *x*- and *y*-axis variable names and with the minimum and maximum values of the display ranges.

Keyboard Access: ← PLOT NXT LABEL

Affected by Flags: None

Remarks: The horizontal axis name is chosen in the following priority order:

1. If the *axes* parameter in the reserved variable *PPAR* is a list, then the *x-axis* element from that list is used.
2. If *axes* parameter is not a list, then the independent variable name in *PPAR* is used.

The vertical axis name is chosen in the following priority order:

1. If the *axes* parameter in *PPAR* is a list, then the *y-axis* element from that list is used.
2. If *axes* is not a list, then the dependent variable name from PPAR is used.

Related Commands: AXES, DRAW, DRAX

LAST

Last Arguments Command: Returns copies of the arguments of the most recently executed command.

Keyboard Access: None. Must be typed in.

Remarks: LAST is provided for compatibility with the HP 28S. LAST is the same as LASTARG. See LASTARG.

LASTARG

Last Arguments Command: Returns copies of the arguments of the most recently executed command.

Level 1	→	Level n	...	Level 1
	→	obj_n	...	obj_1

Keyboard Access:

→ ARG

PRG NXT ERRM ERRN

Affected by Flags: Last Arguments (−55)

Remarks: The objects return to the same stack levels that they originally occupied. Commands that take no arguments leave the current saved arguments unchanged.

When LASTARG follows a command that evaluates an algebraic expression or a program (as do ∂, $\int$, TAYLR, COLCT, DRAW, ROOT, ISOL, EVAL, and →NUM), the last arguments saved are from the evaluated algebraic expression or program, not from the original command.

Related Commands: LAST

LCD→

LCD to Graphics Object Command: Returns the current stack and menu display as a 131 × 64 graphics object.

Level 1	→	Level 1
	→	*grob*

LCD→

Keyboard Access: PRG GRAPH NXT LCD→

Affected by Flags: None

Example: LCD→ PICT STO PICTURE returns the current display to level 1 as a graphics object, stores it in *PICT*, then shows the image in the Picture environment.

Related Commands: →GROB, →LCD

→LCD

Graphics Object to LCD Command: Displays the graphics object from level 1, with its upper left pixel in the upper left corner of the display.

{ }

Level 1	→	Level 1
grob	→	

Keyboard Access: PRG GRAPH NXT →LCD

Affected by Flags: None

Remarks: If the graphics object is larger than 131 × 56, it is truncated.

Related Commands: BLANK, →GROB, LCD→

LIBEVAL

Evaluate Library Function Command: Evaluates unnamed library functions.

{ }

Level 1	→	Level 1
$\#n_{function}$	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Using LIBEVAL with random addresses can corrupt memory. $\#n_{function}$ is of the form *lllfffh*, where *lll* is the library number, and *fff* the function number.

Related Commands: EVAL, SYSEVAL

LIBS

Libraries Command: Lists the title, number, and port of each library attached to the current directory.

Level 1	→	Level 1
	→	{ "*title*" n_{lib} n_{port} ... "*title*" n_{lib} n_{port} }

Keyboard Access: ↰ LIBRARY LIBS

Affected by Flags: None

Remarks: The title of a library often takes the form *LIBRARY-NAME Description*. A library without a title is displayed as " "

Related Commands: ATTACH, DETACH

LINE

Draw Line Command: Draws a line in *PICT* between the coordinates in levels 1 and 2.

Level 2	Level 1	→	Level 1
(x_1, y_1)	(x_2, y_2)	→	
{ # n_1 # m_1 }	{ # n_2 # m_2 }	→	

Keyboard Access: PRG PICT LINE

Affected by Flags: None

Example: This program:

```
« (0,0) (2,3) LINE ( # 0d # 0d ) PVIEW 7 FREEZE »
```

draws a line in *PICT* between two user-unit coordinates, displays *PICT* with pixel coordinate (# 0d # 0d) at the upper left corner of the picture display, and freezes the display.

Related Commands: ARC, BOX, TLINE

ΣLINE

Regression Model Formula Command: Returns an expression representing the best fit line according to the current statistical model, using X as the independent variable name, and explicit values of the slope and intercept taken from the reserved variable ΣPAR.

Level 1	→	Level 1
	→	'$symb_{formula}$'

Keyboard Access: ← STAT FIT ΣLINE

Affected by Flags: None

Remarks: For each curve fitting model, the following table indicates the form of the expression returned by ΣLINE, where m is the slope, x is the independent variable, and b is the intercept.

Model	Form of Expression
LINFIT	$mx + b$
LOGFIT	$m \ln(x) + b$
EXPFIT	be^{mx}
PWRFIT	bx^{m}

Example: If the current model is EXPFIT, and if the slope is 5 and the intercept 3, ΣLINE returns '3*EXP(5*X)'

Related Commands: BESTFIT, COLΣ, CORR, COV, EXPFIT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, XCOL, YCOL

LINFIT

Linear Curve Fit Command: Stores LINFIT as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the linear curve fitting model.

Keyboard Access: ← STAT SFIT MODL LINF

Affected by Flags: None

Remarks: LINFIT is the default specification in ΣPAR. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, EXPFIT, LOGFIT, LR, PWRFIT

LININ

Linear Test Function: Tests whether an algebraic is structurally linear for a given variable.

{}

Level 2	Level 1	→	Level 1
'symb'	*'name'*	→	0/1

Keyboard Access: PRG TEST PREV LININ

Affected by Flags: None

Remarks: If any two subexpressions containing a variable (*name*) are combined only with addition and subtraction, and any subexpression containing the variable is at-most multiplied or divided by another factor not containing the variable, the algebraic (*'symb'*) is determined to be linear for that variable.

LININ returns a 1 if the algebraic is linear for the variable, and a 0 if not.

Example:

```
'(X+1)*(Y^-2^2)+(X/(3-Z^3)'
'X'
LININ
```

returns 1.

```
'(X^2-1)/(X+1)'
'X'
LININ
```

returns 0.

(Although this equation yields a linear equation when factored, LININ tests the equation as described above.)

LIST→

List to Stack Command: Takes a list of n objects and returns them to separate levels, and returns the total number of objects to level 1.

Level 1	→	Level n+1 ...	Level 2	Level 1
{ obj_1 ... obj_n }	→	obj_1 ...	obj_n	n

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: The command OBJ→ also provides this function. LIST→ is included for compatibility with the HP 28S.

Related Commands: ARRY→, DTAG, EQ→, →LIST, OBJ→, STR→

→LIST

Stack to List Command: Takes n objects from level n+1 through level 2 and returns a list of those n objects.

Level n+1 ...	Level 2	Level 1	→	Level 1
obj_1 ...	obj_n	n	→	{ obj_1 ... obj_n }

Keyboard Access: PRG TYPE →LIST

Affected by Flags: None

Example: The program

```
« DEPTH →LIST 'A' STO »
```

combines the entire contents of the stack into a list that is stored in variable A.

→LIST

Related Commands: →ARRY, LIST→, →STR, →TAG, →UNIT

ΣLIST

List Sum Command: Returns the sum of the elements in a list.

Level 1	→	Level 1
{ *list* }	→	*sum*

Keyboard Access: (MTH) LIST ΣLIST

Affected by Flags: None

Remarks: The elements in the list must be suitable for mutual addition.

Examples: { 5 8 2 } ΣLIST returns 15.

{ A B C 1 } ΣLIST returns 'A+B+C+1'

Related Commands: ΠLIST, STREAM

ΔLIST

List Differences Command: Returns the first differences of the elements in a list.

Level 1	→	Level 1
{ *list* }	→	{ *differences* }

Keyboard Access: (MTH) LIST ΔLIST

Affected by Flags: None

Remarks: Adjacent elements in the list must be suitable for mutual subtraction.

Examples: { 4 20 1 17 60 91 } ΔLIST returns { 16 -19 16 43 31 }.

{ A B C 1 2 3 } ΔLIST returns { 'B-A' 'C-B' '1-C' 1 1 }.

{ 'A+3' 'X/5' 'Y^4' } ΔLIST returns
{ 'X/5-(A+3)' 'Y^4-X/5' }.

Related Commands: ΣLIST, ΠLIST, STREAM

ΠLIST

List Product Command: Returns the product of the elements in a list.

Level 1	→	Level 1
{ *list* }	→	*product*

Keyboard Access: (PRG) LIST ΠLIST

Affected by Flags: None

Remarks: The elements in the list must be suitable for mutual multiplication.

Examples: { 5 8 2 } ΠLIST returns 80.

{ A B C 1 } ΠLIST returns 'A*B*C'

Related Commands: ΣLIST, ΔLIST, STREAM

LN

Natural Logarithm Analytic Function: Returns the natural (base e) logarithm of the argument.

{}

Level 1	→	Level 1
z	→	ln z
'symb'	→	'LN(symb)'

Keyboard Access: ↱ LN

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22)

Remarks: For x=0 or (0, 0), an `Infinite Result` exception occurs, or, if flag −22 is set, −MAXR is returned.

The inverse of EXP is a *relation*, not a function, since EXP sends more than one argument to the same result. The inverse relation for EXP is expressed by ISOL as the *general solution*

```
'LN(Z)+2*π*1*n1'
```

The function LN is the inverse of a *part* of EXP, a part defined by restricting the domain of EXP such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of EXP are called the *principal values* of the inverse relation. LN in its entirety is called the *principal branch* of the inverse relation, and the points sent by LN to the boundary of the restricted domain of EXP form the *branch cuts* of LN.

The principal branch used by the HP 48 for LN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued natural log function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

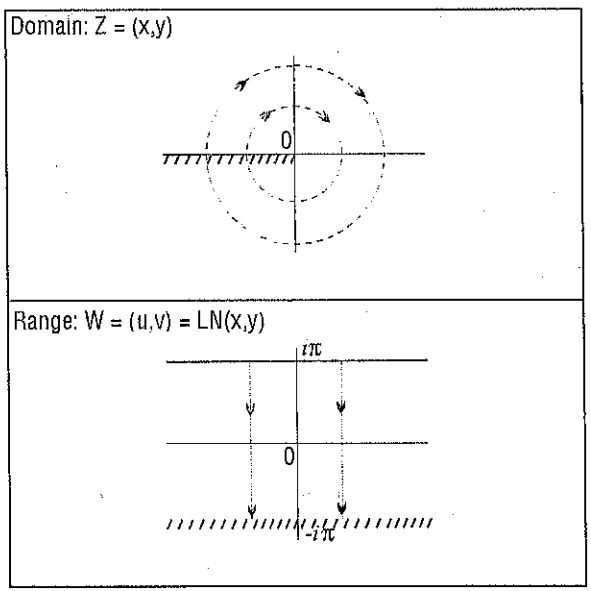
The graphs below show the domain and range of LN. The graph of the domain shows where the branch cut occurs: the heavy solid line marks

one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation `'LN(Z)+2*π*1*n1'` for the case $n1$=0. For other values of $n1$, the horizontal band in the lower graph is translated up (for $n1$ positive) or down (for $n1$ negative). Taken together, the bands cover the whole complex plane, which is the domain of EXP.

You can view these graphs with domain and range reversed to see how the domain of EXP is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain `Z = (x,y)`. EXP sends this domain onto the whole complex plane in the range `W = (u,v) = EXP(x,y)` in the upper graph.

Related Commands: ALOG, EXP, ISOL, LNP1, LOG



LNP1

Natural Log of x Plus 1 Analytic Function: Returns ln $(x + 1)$.

{}

Level 1	→	Level 1
x	→	ln(x+1)
'symb'	→	'LNP1(symb)'

Keyboard Access: MTH HYP NXT LNP1

Affected by Flags: Numerical Results (−3), Infinite Result Exception (−22)

Remarks: For values of x close to zero, 'LNP1(x)' returns a more accurate result than does 'LN(x+1)'. Using LNP1 allows both the argument and the result to be near zero, and it avoids an intermediate result near 1. The calculator can express numbers within 10^{-449} of zero, but within only 10^{-11} of 1.

For values of $x < -1$, an `Undefined Result` error results. For x=−1, an `Infinite Result` exception occurs, or, if flag −22 is set, LNP1 returns −MAXR.

Related Commands: EXPM, LN

LOG

Common Logarithm Analytic Function: Returns the common logarithm (base 10) of the argument.

{}

Level 1	→	Level 1
z	→	log z
'symb'	→	'LOG(symb)'

Keyboard Access: → LOG

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22)

Remarks: For x=0 or (0, 0), an `Infinite Result` exception occurs, or, if flag −22 is set (no error), LOG returns −MAXR.

The inverse of ALOG is a *relation*, not a function, since ALOG sends more than one argument to the same result. The inverse relation for ALOG is expressed by ISOL as the *general solution*

```
'LOG(Z)+2*π*i*ni/2.30258509299'
```

The function LOG is the inverse of a *part* of ALOG, a part defined by restricting the domain of ALOG such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of ALOG are called the *principal values* of the inverse relation. LOG in its entirety is called the *principal branch* of the inverse relation, and the points sent by LOG to the boundary of the restricted domain of ALOG form the *branch cuts* of LOG.

The principal branch used by the HP 48 for LOG(z) was chosen because it is analytic in the regions where the arguments of the real-valued function are defined. The branch cut for the complex-valued LOG function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

You can determine the graph for LOG(z) from the graph for LN (see LN) and the relationship $\log z = \ln z \,/\, \ln 10$.

Related Commands: ALOG, EXP, ISOL, LN

LOGFIT

Logarithmic Curve Fit Command: Stores LOGFIT as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the logarithmic curve-fitting model.

Keyboard Access: ↰ STAT 2-PAR MODL LOGFIT

Affected by Flags: None

Remarks: LINFIT is the default specification in ΣPAR. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, EXPFIT, LINFIT, LR, PWRFIT

LQ

LQ Factorization of a Matrix Command: Returns the LQ factorization of an $n \times m$ matrix.

{}

Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]] A	→	[[*matrix*]]L	[[*matrix*]]Q	[[*matrix*]]P

Keyboard Access: MTH MATR FACTR LQ

Affected by Flags: None

Remarks: LQ factors an $m \times n$ matrix A into three matrices:

- L is a lower $m \times n$ trapezoidal matrix.
- Q is an $n \times n$ orthogonal matrix.
- P is a $m \times m$ permutation matrix.

Where $P \times A = L \times Q$.

Related Commands: LSQ, QR

LR

Linear Regression Command: Uses the currently selected statistical model to calculate the linear regression coefficients (intercept and slope) for the selected dependent and independent variables in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 2	Level 1
	→	Intercept: x_1	Slope: x_2

Keyboard Access: ↰ STAT FIT LR

Affected by Flags: None

Remarks: The columns of independent and dependent data are specified by the first two elements in the reserved variable ΣPAR, set by XCOL and YCOL, respectively. (The default independent and dependent columns are 1 and 2.) The selected statistical model is the fifth element in ΣPAR. LR stores the intercept and slope (untagged) as the third and fourth elements, respectively, in ΣPAR.

The coefficients of the exponential (EXPFIT), logarithmic (LOGFIT), and power (PWRFIT) models are calculated using transformations that allow the data to be fitted by standard linear regression. The equations for these transformations appear in the table below, where b is the intercept and m is the slope. The logarithmic model requires positive x-values (XCOL), the exponential model requires positive y-values (YCOL), and the power model requires positive x- and y-values.

For a description of ΣPAR, see appendix D, "Reserved Variables."

Transformation Equations

Model	Transformation
Logarithmic	$y = b + m \ln(x)$
Exponential	$\ln(y) = \ln(b) + mx$
Power	$\ln(y) = \ln(b) + m \ln(x)$

LR

Related Commands: BESTFIT, COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, PREDX, PREDY, PWRFIT, XCOL, YCOL

LSQ

Least Squares Solution Command: Returns the minimum norm least squares solution to any system of linear equations where $A \times X = B$.

{}

Level 2	Level 1	→	Level 1
[*array*]$_B$	[[*matrix*]]$_A$	→	[*array*]$_X$
[[*matrix*]]$_B$	[[*matrix*]]$_A$	→	[[*matrix*]]$_X$

Keyboard Access:

(↶) (SOLVE) SYS LSQ

(MTH) MATR LSQ

Affected by Flags: Singular Values (−54)

Remarks: If B is a vector, the resulting vector has a minimum Euclidean norm $||X||$ over all vector solutions that minimize the residual Euclidean norm $||A \times X - B||$. If B is a matrix, each column of the resulting matrix, X_i, has a minimum Euclidean norm $||X_i||$ over all vector solutions that minimize the residual Euclidean norm $||A \times X_i - B_i||$.

If A has less than full row rank (the system of equations is underdetermined), an infinite number of solutions exist. LSQ returns the solution with the minimum Euclidean length.

If A has less than full column rank (the system of equations is overdetermined), a solution that satisfies all the equations may not exist. LSQ returns the solution with the minimum residuals of $A \times X - B$.

Related Commands: LQ, RANK, QR, /

LU

LU Decomposition of a Square Matrix Command: Returns the LU decomposition of a square matrix.

{}

Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_L$	[[*matrix*]]$_U$	[[*matrix*]]$_P$

Keyboard Access: (MTH) MATR FACTR LU

Affected by Flags: None

Remarks: When solving an exactly determined system of equations, inverting a square matrix, or computing the determinant of a matrix, the HP 48 factors a square matrix into its Crout LU decomposition using partial pivoting.

The Crout LU decomposition of A is a lower-triangular matrix L, an upper-triangular matrix U with ones on its diagonal, and a permutation matrix P, such that $P \times A = L \times U$. The results satisfy $P \times A \cong L \times U$.

Related Commands: DET, INV, LSQ, /

MANT

Mantissa Function: Returns the mantissa of the argument.

{}

Level 1	→	Level 1
x	→	y_{mant}
'*symb*'	→	'MANT(*symb*)'

Keyboard Access: (MTH) REAL (NXT) MANT

MANT

Affected by Flags: Numerical Results (−3)

Example: `-1.2E34 MANT` returns `1.2`.

Related Commands: SIGN, XPON

↑MATCH

Bottom-Up Match and Replace Command: Rewrites an expression.

Level 2	Level 1	→	Level 2	Level 1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' }	→	'$symb_2$'	0/1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' '$symb_{cond}$' }	→	'$symb_2$'	0/1

Keyboard Access: ↰ SYMBOLIC NXT ↑MAT

Affected by Flags: None

Remarks: ↑MATCH rewrites expressions or subexpressions that match a specified pattern '$symb_{pat}$'. An optional condition, '$symb_{cond}$', can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; `1` if it did, `0` if it did not.

The pattern '$symb_{pat}$' and replacement '$symb_{repl}$' can be normal expressions; for example, you can replace `'SIN(π/6)'` with `'1/2'`. You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with `&`, such as the name `'&A'`, used in replacing `'SIN(&A+π)'` with `'-SIN(&A)'`. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↑MATCH works from bottom up; that is, it checks the lowest level (most deeply nested) subexpressions first. This approach works well for simplification. A subexpression simplified during one execution of ↑MATCH will be a simpler argument of its parent expression, so the parent expression can be simplified by another execution of ↑MATCH.

Several subexpressions can be simplified by one execution of ↑MATCH provided none is a subexpression of any other.

Examples: This sequence:

```
'SIN(π/6)' { 'SIN(π/6)' '1/2' } ↑MATCH
```

returns `'1/2'` to level 2 and `1` (indicating a replacement was made) to level 1.

This sequence:

```
'SIN(X+π)' { 'SIN(&A+π)' '-SIN(&A)' } ↑MATCH
```

returns `'-SIN(X)'` to level 2 and `1` to level 1.

This sequence:

```
'W+√(SQ(5))' { '√(SQ&A))' '&A' '&A≥0' } ↑MATCH
```

returns `'W+5'` to level 2 and `1` to level 1.

Related Commands: ↓MATCH

↓MATCH

Match Pattern Down Command: Rewrites an expression.

Level 2	Level 1	→	Level 2	Level 1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' }	→	'$symb_2$'	0/1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' '$symb_{cond}$' }	→	'$symb_2$'	0/1

Keyboard Access: ↰ SYMBOLIC NXT ↓MAT

Affected by Flags: None

Remarks: ↓MATCH rewrites expressions or subexpressions that match a specified pattern '$symb_{pat}$'. An optional condition, '$symb_{cond}$', can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; `1` if it did, `0` if it did not.

↓MATCH

The pattern '$symb_{pat}$' and replacement '$symb_{repl}$' can be normal expressions; for example, you can replace .5 with 'SIN(π/6)'. You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with &, such as the name '&A', used in replacing 'SIN(&A+&B)' with 'SIN(&A)*COS(&B)+COS(&A)*SIN(&B)'. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↓MATCH works from top down; that is, it checks the entire expression first. This approach works well for expansion. An expression expanded during one execution of ↓MATCH will contain additional subexpressions, and those subexpressions can be expanded by another execution of ↓MATCH. Several expressions can be expanded by one execution of ↓MATCH provided none is a subexpression of any other.

Examples:

```
.5 « .5 'SIN(π/6)' » ↓MATCH
```

returns 'SIN(π/6)' to level 2 and 1 to level 1.

```
'SIN(U+V)' « 'SIN(&A+&B)'
'SIN(&A)*COS(&B)+COS(&A)*SIN(&B)' » ↓MATCH
```

returns 'SIN(U)*COS(V)+COS(U)*SIN(V)' to level 2 and 1 to level 1.

This sequence:

```
'SIN(5*Z)' « 'SIN(&A+&B)'
'Σ(K=0,&A,COMB(&A,K)*SIN(K*π)*
COS(&B^(&A-K)*SIN(&B)^K)'
'ABS(IP(&A))==&A' » ↓MATCH
```

returns

```
'Σ(K=0,5,COMB(5,K)*SIN(K*π)*COS(Z)^(5-K)*SIN(Z)^K)'
```

to level 2 and 1 to level 1.

Related Commands: ↑MATCH

MAX

Maximum Function: Returns the greater (more positive) of the arguments.

{}

Level 2	Level 1	→	Level 1
x	y	→	max(x, y)
x	'$symb$'	→	'MAX(x, $symb$)'
'$symb$'	x	→	'MAX($symb$, x)'
'$symb_1$'	'$symb_2$'	→	'MAX($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	max(x_unit_1, y_unit_2)

Keyboard Access: (MTH) REAL MAX

Affected by Flags: Numerical Results (−3)

Examples: 10 −23 MAX returns 10.

−10 −23 MAX returns −10.

i_m 9_cm MAX returns i_m.

Related Commands: MIN

MAXR

Maximum Real Function: Returns the symbolic constant 'MAXR' or its numerical representation, 9.99999999999E499.

Level 1	→	Level 1
	→	'MAXR'
	→	9.99999999999E499

Keyboard Access: (MTH) (NXT) CONS (NXT) MAXR

MAXR

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

MAXR returns its numerical representation if flag −2 or −3 is set; otherwise, it returns its symbolic representation.

Remarks: MAXR is the largest numerical value that can be represented by the HP 48.

Related Commands: e, i, MINR, π

MAXΣ

Maximum Sigma Command: Finds the maximum coordinate value in each of the m columns of the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{max}
	→	$[x_{max1}\ x_{max2}\ \cdots\ x_{maxm}]$

Keyboard Access: ◀ STAT MAXΣ

Affected by Flags: None

Remarks: The maxima are returned as a vector of m real numbers, or as a single real number if $m = 1$.

Related Commands: BINS, MEAN, MINΣ, SDEV, TOT, VAR

MCALC

Make Calculated Value Command: Designates a variable as a calculated value (not user-defined) for the Multiple-Equation Solver.

Level 1	→	Level 1
'name'	→	
{ *list* }	→	
"ALL"	→	

Keyboard Access: ◀ EQ LIB MES MCAL

Affected by Flags: None

Remarks: MCALC designates a single variable, a list of variables, or all variables as calculated values.

Related Commands: MUSER

MEAN

Mean Command: Returns the mean of each of the m columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{mean}
	→	$[\ x_{mean1}\ x_{mean2}\ \cdots\ x_{meanm}\ ]$

Keyboard Access: ◀ STAT MAΣ MEAN

Affected by Flags: None

MEAN

Remarks: The mean is returned as a vector of m real numbers, or as a single real number if $m = 1$. The mean is computed from the formula:

$$\frac{1}{n}\sum_{i=1}^{n} x_i$$

where x_i is the ith coordinate value in a column, and n is the number of data points.

Related Commands: BINS, MAXΣ, MINΣ, SDEV, TOT, VAR

MEM

Memory Available Command: Returns the number of bytes of available RAM.

Level 1	→	Level 1
	→	*x*

Keyboard Access: ◀ MEMORY MEM

Affected by Flags: None

Remarks: The number returned is only a rough indicator of usable available memory, since recovery features (LASTARG, ▶ UNDO, and ▶ CMD) consume or release varying amounts of memory with each operation.

Before it can assess the amount of memory available, MEM must remove objects in temporary memory that are no longer being used. This clean-up process (also called "garbage collection") also occurs automatically at other times when memory is full. Since this process can slow down calculator operation at undesired times, you can force it to occur at a desired time by executing MEM. In a program, execute MEM DROP

Related Commands: BYTES

MENU

Display Menu Command: Displays a built-in menu or a library menu, or defines and displays a custom menu.

Level 1	→	Level 1
x_{menu}	→	
{ $list_{definition}$ }	→	
'$name_{definition}$'	→	
obj	→	

Affected by Flags: None

Remarks: A built-in menu is specified by a real number x_{menu}. The format of x_{menu} is $mm.pp$, where mm is the menu number and pp is the page of the menu. If pp doesn't correspond to a page of the specified menu, the first page is displayed. The following table lists the HP 48 built-in menus and the corresponding menu numbers.

Menu #	Menu Name	Menu #	Menu Name
0	Last Menu	15	MTH BASE
1	CST	16	MTH BASE LOGIC
2	VAR	17	MTH BASE BIT
3	MTH	18	MATH BASE BYTE
4	MTH VECTR	19	MTH FFT
5	MTH MATR	20	MTH CMPL
6	MTH MATR MAKE	21	MTH CONS
7	MTH MATR NORM	22	PRG
8	MTH MATR FACTR	23	PRG BRCH
9	MTH MATR COL	24	PRG BRCH IF
10	MTH MATR ROW	25	PRG BRCH CASE
11	MTH LIST	26	PRG BRCH START
12	MTH HYP	27	PRG BRCH FOR
13	MTH PROB	28	EDIT
14	MTH REAL	29	PRG BRCH DO

Menu #	Menu Name
30	SOLVE ROOT SOLVR
31	PRG BRCH WHILE
32	PRG TEST
33	PRG TYPE
34	PRG LIST
35	PRG LIST ELEM
36	PRG LIST PROC
37	PRG GROB
38	PRG PICT
39	PRG IN
40	PRG OUT
41	PRG RUN
42	UNITS (Units Catalog Menu)
43	UNITS LENG
44	UNITS AREA
45	UNITS VOL
46	UNITS TIME
47	UNITS SPEED
48	UNITS MASS
49	UNITS FORCE
50	UNITS ENRG
51	UNITS POWR
52	UNITS PRESS
53	UNITS TEMP
54	UNITS ELEC
55	UNITS ANGL
56	UNITS LIGHT
57	UNITS RAD
58	UNITS VISC
59	UNITS
60	PRG ERROR IFERR
61	PRG ERROR

Menu #	Menu Name
62	CHARS
63	MODES
64	MODES FMT
65	MODES ANGL
66	MODES FLAG
67	MODES KEYS
68	MODES MENU
69	MODES MISC
70	MEMORY
71	MEMORY DIR
72	MEMORY ARITH
73	STACK
74	SOLVE
75	SOLVE ROOT
76	SOLVE DIFFE
77	SOLVE POLY
78	SOLVE SYS
79	SOLVE TVM
80	SOLVE TVM SOLVR
81	PLOT
82	PLOT PTYPE
83	PLOT PPAR
84	PLOT 3D
85	PLOT 3D PTYPE
86	PLOT 3D VPAR
87	PLOT STAT
88	PLOT STAT PTYPE
89	PLOT STAT ΣPAR
90	PLOT STAT ΣPAR MODL
91	PLOT STAT DATA
92	PLOT FLAG
93	SYMBOLIC

Menu #	Menu Name
94	TIME
95	TIME ALARM
96	STAT
97	STAT DATA
98	STAT ΣPAR
99	STAT ΣPAR MODL
100	STAT 1VAR
101	STAT PLOT
102	STAT FIT
103	STAT SUMS
104	IO
105	IO SRVR
106	IO IOPAR

Menu #	Menu Name
107	IO PRINT
108	IO PRINT PRTPA
109	IO SERIA
110	LIBRARY
111	LIBRARY PORTS
112	LIBRARY PORTS :0:
113	EQ LIB
114	EQ LIB EQLIB
115	EQ LIB COLIB
116	EQ LIB MES
117	EQ LIB UTILS

Library menus are specified in the same way as built-in menus, with the library number serving as the menu number.

Custom menus are specified by a list of the form `{ "label-object" action-object }` (see appendix D, "Reserved Variables," for details) or a name containing a list (`'name`$_{\text{definition}}$`'`). Either argument is stored in reserved variable *CST*, and the custom menu is subsequently displayed.

MENU takes *any* object as a valid argument and stores it in *CST*. However, the calculator can build a custom menu *only* if *CST* contains a list or a name containing a list. Thus, if an object other than a list or name containing a list is supplied to MENU, a `Bad Argument Type` error will occur when the calculator attempts to display the custom menu.

Examples: `5 MENU` displays the first page of the MTH MATR NORM menu.

`48.02 MENU` displays the second page of the UNITS MASS menu.

`{ A 123 "ABC" } MENU` displays the custom menu defined by the list argument.

`'MYMENU' MENU` displays the custom menu defined by the name argument.

MENU

Related Commands: RCLMENU, TMENU

MERGE

Merge RAM Card Command: Merges the RAM from the card in port 1 with the rest of main user memory. Merged memory is no longer independent.

Level 1	→	Level 1
1	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: MERGE is provided for compatibility with the HP 48S series. See MERGE1.

Related Commands: FREE1, MERGE1

MERGE1

Merge RAM Card Command: Merges the RAM from the card in port 1 with the rest of main user memory. Merged memory is no longer independent.

Keyboard Access: ◀ LIBRARY MERGE

Affected by Flags: None

Remarks: If the RAM card contains library or backup objects, they are moved to port 0 before the RAM is merged. Library and backup objects can exist only in independent memory (port 1 through 33 unmerged, or port 0).

Cards larger than 128K cannot be merged, and cannot be plugged into port 1.

Related Commands: FREE, FREE1, MERGE

MIN

Minimum Function: Returns the lesser (more negative) of its two arguments.

{}

Level 2	Level 1	→	Level 1
x	*y*	→	min(*x*, *y*)
x	'*symb*'	→	'MIN(*x*, *symb*)'
'*symb*'	*x*	→	'MIN(*symb*, *x*)'
'$symb_1$'	'$symb_2$'	→	'MIN($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	min(x_unit_1, y_unit_2)

Keyboard Access: MTH REAL MIN

Affected by Flags: Numerical Results (−3)

Examples: `10 23 MIN` returns `10`.

`-10 -23 MIN` returns `-23`.

`1_m 9_cm MIN` returns `9_cm`.

Related Commands: MAX

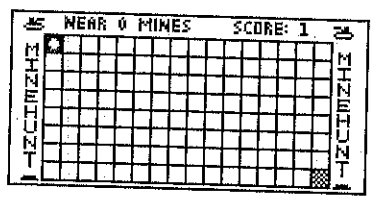
MINEHUNT

MINEHUNT Game Command: Starts the MINEHUNT game.

Keyboard Access: ◀ EQ LIB UTILS MINE

Affected by Flags: None

Remarks: In the game, you are standing in the upper-left corner of an 8 × 16 battlefield grid. Your mission is to travel safely to the lower-right corner, avoiding invisible mines along the way. The game tells you how many mines are under the eight squares adjacent to your position.



Use the number or arrow keys to cross the battlefield one square at a time. (Use 7, 9, 1, and 3 to move diagonally.) You can exit the game any time by pressing CANCEL.

To interrupt and save a game, press STO. This creates a variable *MHpar* in the current directory and ends the game. If *MHpar* exists when you next start Minehunt, the interrupted game resumes and *MHpar* is purged.

You can change the number of mines in the battlefield by creating a variable named *Nmines* containing the desired number. *Nmines* must contain a real number (1 to 64). If *Nmines* is negative, the mines are visible during the game.

MINIT

Multiple Equation Menu Initialization Command: Creates the reserved variable *Mpar*.

Keyboard Access: ◀ EQ LIB MES MINIT

Affected by Flags: None

Remarks: MINIT takes multiple equations stored in *EQ* and creates the multiple equation reserved variable *Mpar*. See appendix D, "Reserved Variables," for information about *Mpar*.

Related Commands: MITM, MROOT, MSOLVR

MINR

Minimum Real Function: Returns the symbolic constant 'MINR' or its numerical representation, 1.0000000000E-499.

Level 1	→	Level 1
	→	'MINR'
	→	1.00000000000E-499

Keyboard Access: MTH NXT CONS NXT MINR

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

MAXR returns its numerical representation if flag −2 or −3 is set; otherwise, it returns its symbolic representation.

Remarks: MINR is the smallest nonzero numerical value that can be represented by the HP 48.

Related Commands: e, i, MAXR, π

MINΣ

Minimum Sigma Command: Finds the minimum coordinate value in each of the m columns of the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{min}
	→	$[\, x_{min1}\ x_{min2}\ \cdots\ x_{minm}\,]$

Keyboard Access: ↰ STAT ΣMTR MTH2

Affected by Flags: None

Remarks: The minima are returned as a vector of m real numbers, or as a single real number if $m = 1$.

Related Commands: BINS, MAXΣ, MEAN, SDEV, TOT, VAR

MITM

Multiple Equation Menu Item Order Command: Changes multiple equation menu titles and order.

Level 2	Level 1	→	Level 1
"title"	*{ list }*	→	

Keyboard Access: ↰ EQ LIB MENU MITM

Affected by Flags: None

Remarks: *list* contains the variable names in the order you want. Use "" to indicate a blank label. You must include *all* variables in the original menu and no others.

Related Commands: MINIT

MOD

Modulo Function: Returns a remainder defined by:

$$x \bmod y = x - y \text{ floor } (x/y)$$

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	*x* mod *y*
x	*'symb'*	→	*'MOD(x, symb)'*
'symb'	*x*	→	*'MOD(symb, x)'*
'symb$_1$'	*'symb$_2$'*	→	*'MOD(symb$_1$, symb$_2$)'*

Keyboard Access: MTH REAL MOD

Affected by Flags: Numerical Results (−3)

Remarks: Mod (x, y) is periodic in x with period y. Mod (x, y) lies in the interval $[0, y)$ for $y > 0$ and in $(y, 0]$ for $y < 0$.

Related Commands: FLOOR, /

MROOT

Multiple Roots Command: Uses the Multiple-Equation Solver to solve for one or more variables using the equation set in *Mpar*.

{ }

Level 1	→	Level 1
'name'	→	*x*
"ALL"	→	

MROOT

Keyboard Access: ◀ EQ LIB MES MROOT

Affected by Flags: None

Remarks: Solves for one or more variables starting with only user-defined values, and leaves found values in the variables. No status messages are displayed. Given a variable name, MROOT returns the found value; it can also take "ALL" (stores a found value for each variable) and return nothing to the stack.

Related Commands: MCALC, MUSER

MSGBOX

Message Box Command: Creates a user-defined message box.

{ }

Level 1	→	Level 1
"*message*"	→	

Keyboard Access: PRG NXT OUT MSGBX

Affected by Flags: None

Remarks: MSGBOX displays "*message*" in the form of a standard message box. Message text longer than 75 characters (including spaces) is truncated to 75 characters. You can use spaces and new-line characters (▶ ↵) to control word-wrapping and line breaks within the message.

Program execution resumes when the message box is exited by selecting OK or CANCEL.

Related Commands: CHOOSE, INFORM, PROMPT

MSOLVR

Multiple-Equation Solver Command: Gets the Multiple-Equation Solver variable menu for the set of equations defined by *Mpar*.

Keyboard Access:

◀ EQ LIB MES MSOL

◀ EQ LIB EQLIB MSOL

Affected by Flags: None

Remarks: The Multiple-Equation Solver application can solve a set of two or more equations for unknown variables by finding the roots of each equation, one at a time.

The Multiple-Equation Solver uses the list of equations stored in *EQ*. "Equations" in this context includes programs, expressions, and variable names that evaluate to a single value. The Multiple-Equation Solver requires that *EQ* contain more than one equation—that is, the HP Solve application would include the NUMER menu label for *EQ*. The solver uses *EQ* to create a reserved variable *Mpar* that is used during the solution process. *Mpar* contains the equation set plus additional information. See appendix D, "Reserved Variables," for information about *Mpar*.

Related Commands: EQNLIB, SOLVEQN

MUSER

Make User-Defined Variable Command: Designates a variable as user-defined for the Multiple-Equation Solver.

Level 1	→	Level 1
'*name*'	→	
{ *list* }	→	
"*ALL*"	→	

MUSER

Keyboard Access: ↰ EQ LIB MTES MUSE

Affected by Flags: None

Remarks: MUSER designates a single variable, a list of variables, or all variables as user-defined.

Related Commands: MCALC

NDIST

Normal Distribution Command: Returns the normal probability distribution (bell curve) at x based on the mean m and variance v of the normal distribution.

{}

Level 3	Level 2	Level 1	→	Level 1
m	*v*	*x*	→	*ndist(m,v,x)*

Keyboard Access: MTH NXT PROB NXT NDIST

Affected by Flags: None

Remarks: NDIST is calculated using this formula:

$$ndist(m, v, x) = \frac{e^{-\frac{(x-m)^2}{2v}}}{\sqrt{2\pi v}}$$

Related Commands: UTPN

NEG

Negate Analytic Function: Changes the sign or negates an object.

{}

Level 1	→	Level 1
z	→	$-z$
$\#n_1$	→	$\#n_2$
[*array*]	→	[*−array*]
'*symb*'	→	'*−(symb)*'
x_unit	→	*−x_unit*
$grob_1$	→	$grob_2$
$PICT_1$	→	$PICT_2$

Keyboard Access:

+/−

MTH NXT CMPL NXT NEG

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: Negating an array creates a new array containing the negative of each of the original elements. Negating a binary number takes its two's complement (complements each bit and adds 1).

Negating a graphics object "inverts" it (toggles each pixel from on to off, or vice-versa). If the argument is *PICT*, the graphics object stored in *PICT* is inverted.

Related Commands: ABS, CONJ, NOT, SIGN