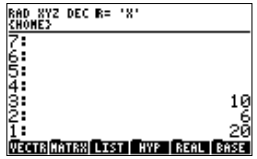
For example, if the stack looks like this:



then

- → a creates local variable *a* = 20.
- → ab creates local variables *a* = 6 and *b* = 20.
- → abc creates local variables *a* = 10, *b* = 6, and *c* = 20.

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

Local variable structures have these advantages:

- The → command stores the values from the stack in the corresponding variables — you don't need to explicitly execute STO.
- Local variables automatically disappear when the defining procedure for which they are created has completed execution. Consequently, local variables don't appear in the VAR menu, and they occupy user memory only during program execution.
- Local variables exist only within their defining procedure — different local variable structures can use the same variable names without conflict.

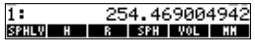
Example: The following program *SPHLV* calculates the volume of a spherical cap using local variables. The defining procedure is an algebraic expression.

Level 2	Level 1	→	Level 1
r	*h*	→	*volume*

Program:	Comments:
«	
→ r h	Creates local variables r and h for the radius of the sphere and height of the cap.
'1/3*π*h^2*(3*r-h)'	Expresses the defining procedure. In this program, the defining procedure for the local variable structure is an algebraic expression.
→NUM	Converts expression to a number.
»	
ENTER ' SPHLV STO▸	Stores the program in variable *SPHLV*.

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

10 ENTER 3



VAR SPHLV

HP 50g / 49g+ / 48gII graphing calculator

advanced user's reference manual

invent

Edition 2

HP part number F2228-90010

Printed Date: 2009/7/14

This is the program:

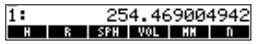
```
« '1/3*π*H^2*(3*R-H)' →NUM »
```

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

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

10 ' R STO

3 ' H STO



VAR

Viewing and Editing Programs

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

To view or edit a program:

1. View the program:
   - If the program is in level 1, press ▼ (or use the EDIT command).
   - If the program is stored in a variable, use the Filer (FILES) to select the variable and press EDIT (F1 A), or press VAR, then ⌐ and the variable's menu key (a shortcut to recall a variable's contents to level 1), followed by ▼. Alternatively, with the variable *name* in level 1 press ◄ ▼ (or use the EDITB, VISIT or VISITB command).
2. Optional: Make changes.
3. Press ENTER to save any changes (or press CANCEL to discard changes) and return to the stack, or to Filer if you used Filer to select the program.

Filer lets you change a stored program without having to do a store operation. From the stack you can change a program and then store the new version in a different variable.

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

To switch between entry modes:

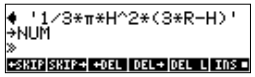
■ Press ⌐ ENTRY .

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

Select *SPH* from the soft keys.

VAR

⌐ EDIT ▼



Move the cursor past the first program delimiter and insert the new program steps.

▶ ' H ▶ STO

' R ▶ STO

Notice

REGISTER YOUR PRODUCT AT: www.register.hp.com

THIS MANUAL AND ANY EXAMPLES CONTAINED HEREIN ARE PROVIDED "AS IS" AND ARE SUBJECT TO CHANGE WITHOUT NOTICE. HEWLETT-PACKARD COMPANY MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

HEWLETT-PACKARD CO. SHALL NOT BE LIABLE FOR ANY ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MANUAL OR THE EXAMPLES CONTAINED HEREIN.

Hewlett-Packard Company
4995 Murphy Canyon Rd,
Suite 301
San Diego, CA 92123

Acknowledgements

Hewlett-Packard would like to thank the following for their contribution:
Jordi Hidalgo, Joe Horn, Tony Hutchins, Ted Kerber, Wlodek Mier-Jedrzejowicz, Richard Nelson, Eric Rechlin, Jake Schwartz and Gene Wright.

Printing History

Edition 1 September 2005
Edition 2 July 2009

To stop an executing program:

■ Press CANCEL .

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

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

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

3 y^x ← π ÷ × 4 ENTER 3 ÷ × → →NUM

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

→ « »

3 y^x ← π ÷ × 4 SPC 3 ÷ ×

→ ← → →NUM

Put the program on the stack.

ENTER

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

' VOL STO▸

4 VAR [VOL]

The program is

« 3 ^ π * 4 3 / * →NUM »

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

« → r '4/3*π*r^3' →NUM »

(You need to include →NUM because π causes a symbolic result, unless Flag –2 or Flag –3 is set)

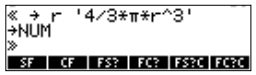
Enter the program. (→ ← just starts a new line.)

→ « »

→ → r SPC

' 4 ÷ 3 × ← π ×

r y^x 3 ▶ → ← → →NUM



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

' VOL STO▸

4 [VOL]

Contents

Examples of Program Actions

Program	Results
`« 1 2 »`	`2: 1` `1: 2`
`« "Hello" { A B } »`	`2: "Hello"` `1: { A B }`
`« '1+2' »`	`1: '1+2'`
`« '1+2' →NUM »`	`1: 3`
`« « 1 2 + » »`	`1: « 1 2 + »`
`« « 1 2 + » EVAL »`	`1: 3`

Programs can also contain *structures*. A structure is a program segment with a defined organization. Two basic kinds of structure are available:

- **Local variable structure.** The → command defines local variable names and a corresponding algebraic or program object that's evaluated using those variables.
- **Branching structures.** Structure words (like DO…UNTIL…END) define conditional or loop structures to control the order of execution within a program.

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

« → *name*$_1$ … *name*$_n$ '*algebraic*' »
« → *name*$_1$ … *name*$_n$ « *program* » »

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

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

```
« → a b 'ABS(a-b)' »
```

Calculations in a Program

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

- **Stack commands.** Operate directly on the objects on the stack.
- **Local variable structures.** Store the stack objects in temporary local variables, then use the variable names to represent the data in the following algebraic or program object.

Numeric calculations provide convenient examples of these methods. The following programs use two numbers from the stack to calculate the hypotenuse of a right triangle using the formula $\sqrt{x^2+y^2}$.

```
« SQ SWAP SQ + √ »
« → x y « x SQ y SQ + √ » »
« → x y '√(x^2+y^2)' »
```

The first program uses stack commands to manipulate the numbers on the stack — the calculation uses stack syntax. The second program uses a local variable structure to store and retrieve the numbers — the calculation uses stack syntax. The third program also uses a local variable structure — the calculation uses algebraic syntax. Note that the underlying formula is most apparent in the third program. This third method is often the easiest to write, read, and debug.

(The ↵ in the previous program is the calculator's representation for the newline character after you enter a program on the stack.)

Using INPUT, ENTER for Input

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

To enter INPUT in a program:

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

« … "*prompt-string*" "*command-line*" INPUT OBJ→ … »

or

« … "*prompt-string*" { *command-line*" INPUT OBJ→ … »

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

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

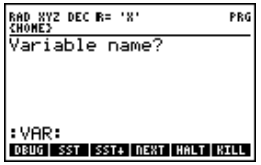
To respond to INPUT while running a program

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

If you execute this program segment

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

the display looks like this:



Evaluating Local Names

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

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

Defining the Scope of Local Variables

Local variables exist *only* inside the defining procedure.

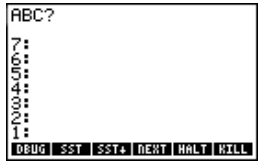
Example: The following program excerpt illustrates the availability of local variables in *nested* defining procedures (procedures within procedures). Because local variables *a*, *b*, and c already exist when the defining procedure for local variables *d*, *e*, and *f* is executed, they're available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

Program:	Comments:
«	
.	
.	
.	No local variables are available.
→ a b c	Defines local variables *a*, *b*, *c*.
«	Local variables *a*, *b*, *c* are available
a b + c +	in this procedure.
→ d e f	Defines local variables *d*, *e*, *f*.
'a/(d*e+f)'	Local variables *a*, *b*, *c* and *d*, *e*, *f* are available in this procedure.
a c / −	Only local variables *a*, *b*, *c* are available.
»	
.	No local variables are available.
.	
.	
»	

Example: If you execute this program segment

```
« "ABC?" PROMPT »
```

the display looks like this:

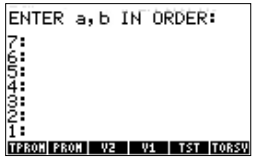


Example: The following program, *TPROMPT*, prompts you for the dimensions of a torus, then calls program *TORSA* (from page 1-29) to calculate its surface area. You don't have to enter data on the stack prior to program execution.

Program:	Comments:
« "ENTER a, b IN ORDER:" PROMPT TORSA »	Puts the prompting string on the stack. Displays the string in the status area, halts program execution, and returns calculator control to the keyboard. Executes *TORSA* using the just-entered stack arguments.
ENTER ' TPROMPT STO▶	Stores the program in *TPROMPT*.

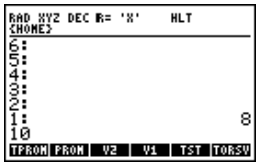
Execute TPROMPT to calculate the volume of a torus with inner radius a = 8 and outer radius b = 10. Execute TPROMPT. The program prompts you for data.

▱ CLEAR VAR TPROM



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

8 ENTER 10



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

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

Using Tests and Conditional Structures

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

Testing Conditions

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

To include a test in a program:

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

You often use test results in conditional structures to determine which clause of the structure to execute. Conditional structures are described under Using Conditional Structures and Commands, page 1-13.

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

Using Comparison Functions

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

Comparison Functions

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

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

All comparison commands except SAME return the following:

The IFERR … THEN … ELSE … END Structure

The syntax for this structure is

« … IFERR *trap-clause*
THEN *error-clause* ELSE *normal-clause* END … »

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

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

■ Press ◁ PRG NXT NXT PROG ▷ BRCH

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

Program:	Comments:
«	
DO	Begins the main loop.
"KEY IN a AND b" " " INPUT OBJ→	Prompts for two numbers.
UNTIL	Starts the loop test clause.
IFERR +	The error trap contains only the + command.
THEN ERRM 5 DISP 2 WAIT 0	If an error occurs, recalls and displays the Too Few Arguments message for 2 seconds, then puts 0 (false) on the stack for the main loop.
ELSE 1	If no error occurs, puts 1 (true) on the stack for the main loop.
END	Ends the error trap.
END	Ends the main loop. If the error trap left 0 (false) on the stack, the main loop repeats — otherwise, the program ends.
»	

Testing Object Types

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

Testing Linear Structure

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

Using Conditional Structures and Commands

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

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

These conditional structures and commands are contained in the PRG BRCH menu (◁ PRG BRCH):

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

The IF … THEN … END Structure

The syntax for this structure is

« … IF *test-clause* THEN *true-clause* END … »

IF … THEN … END executes the sequence of commands in the *true-clause* only if the test-clause evaluates to true. The *test-clause* can be a command sequence (for example, A B ≤) or an algebraic (for example, 'A≤B'). If the test-clause is an algebraic, it's *automatically evaluated* to a number — you don't need →NUM or EVAL.

IF begins the test-clause, which leaves a test result on the stack. THEN removes the test result from the stack. If the value is nonzero, the true-clause is executed — otherwise, program execution resumes following END. See "Conditional Examples" on page 1-15.

To enter IF … THEN … END in a program:

■ Press ◁ PRG BRCH ◁ IF .

The IFT Command

The IFT command takes two arguments: a *test-result* in level 2 and a *true-clause* object in level 1. If the test-result is true, the true-clause object is executed — otherwise, the two arguments are removed from the stack. See "Conditional Examples" on page 1-15.

To enter IFT in a program:

■ Press ◁ PRG BRCH NXT ◁ IFT .

The IF … THEN … ELSE … END Structure

The syntax for this structure is

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

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

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

To analyze an error in a program:

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

The error number for a user-defined error is #70000h. See the list of built-in error numbers in appendix A, "Error and Status Messages".

Example: The following program aborts execution if the list in level 1 contains three objects.

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

The following table summarizes error trapping commands.

Error Trapping Commands

Key	Programmable Command	Description
← PRG NXT NXT ERROR:		
DOERR	DOERR	Causes an error. For a string in level 1, causes a user-defined error: the calculator behaves just as if an ordinary error has occurred. For a binary integer or real number in level 1, causes the corresponding built-in error. If the error isn't trapped in an IFERR structure, DOERR displays the message and abandons program execution. (For 0 in level 1, abandons execution without updating the error number or message — like CANCEL.)
ERRN	ERNN	Returns the error number, as a binary integer, of the most recent error. Returns #0 if the error number was cleared by ERR0.
ERRM	ERRM	Returns the error message (a string) for the most recent error. Returns an empty string if the error number was cleared by ERR0.
ERR0	ERR0	Clears the last error number and message.

To enter CASE … END in a program:

1. Press ← PRG CASE ← IFTE to enter CASE … THEN …END…END
2. For each additional test-clause, move the cursor after a test-clause END and press → IFTE to enter THEN … END.

Conditional Examples

These examples illustrate conditional structures in programs.

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

```
« DUP IF 0 > THEN NEG END »
```

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

The following version uses an algebraic as the test clause:

```
« → x « x IF 'x>0' THEN NEG END » »
```

The following version uses the IFT command:

```
« DUP 0 > « NEG » IFT »
```

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

Program:	Comments:
«	
→ x y	Creates local variables x and y containing the two numbers from the stack.
«	
IF	Starts the test-clause.
'x≠0'	Tests one of the numbers and leaves a test result on the stack.
'y≠0'	Tests the other number, leaving another test result on the stack.
AND	Tests whether both tests were true.
THEN	Ends the test-clause, starts the true-clause.
x y *	Multiplies the two numbers together only if AND returns true.
END	Ends the true-clause.
»	
»	

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

```
« → x y « IF 'x AND y' THEN x y * END » »
```

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

The following version uses the IFT command:

```
« → x y « 'x AND y' 'x*y' IFT » »
```

To single-step from the middle of a program:

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

When you want the program to run normally again, remove the HALT command from the program.

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

- To execute the subroutine in one step ("step over"), press SST.
- To execute the subroutine step-by-step ("step into"), press SST↓.

SST executes the next step in a program — if the next step is a subroutine, SST executes the subroutine in one step. SST↓ works just like SST — except if the next program step is a subroutine, it single-steps to the first step in the subroutine.

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

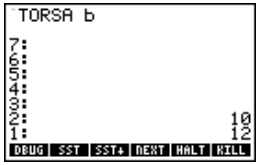
Select the VAR menu and enter the data. Enter the program name and start the debugging. Execute the first four steps of the program, then check the next step.

← CLEAR VAR 10 ENTER 12 ' TORSV

← PRG NXT NXT RUN DEBUG

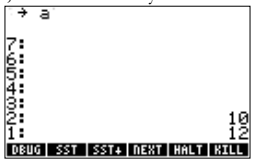
SST↓ (4 times)

NEXT



The next step is TORSA. Single-step into *TORSA*, then check that you're at the first step of *TORSA*.

SST↓ NEXT



Press ← CONT ← CONT to complete subroutine and program execution. The following table summarizes the operations for single-stepping through a program.

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

Program:	Comments:
«	
→ y	Defines local variable *y*.
«	Starts the defining procedure.
CASE	Starts the case structure.
y TYPE 2 SAME THEN y 'STR' STO END	Case 1: If the argument is a string, stores it in *STR*.
y TYPE 5 SAME THEN y 'LIST' STO END	Case 2: If the argument is a list, stores it in *LIST*.
y TYPE 8 SAME THEN y 'PROG' STO END	Case 3: If the argument is a program, stores it in *PROG*.
END	Ends the case structure.
»	Ends the defining procedure.
»	

Using Loop Structures

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

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

- START … NEXT and START … STEP.
- FOR … NEXT and FOR … STEP
- DO … UNTIL … END.
- WHILE … REPEAT … END.

In addition, the Σ function provides an alternative to definite loop structures for summations.

Using Definite Loop Structures

Each of the two definite loop structures has two variations:

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

Program:	Comments:
«	
→ a b	Creates local variables a and b.
'π^2*(b^2-a^2)'	Calculates the surface area.
→NUM	Converts algebraic to a number.
»	
ENTER ' TORSA STO▸	Puts the program on the stack. Stores the program in *TORSA*.

Here is a stack diagram and program listing for *TORSV*.

Level 2	Level 1	→	Level 1
a	***b***	→	***volume***

Program:	Comments:
«	
→ a b	Creates local variables *a* and *b*.
«	Starts a program as the defining procedure.
a b TORSA	Puts the numbers stored in *a* and *b* on the stack, then calls *TORSA* with those arguments.
b a - * 4 /	Completes the volume calculation using the surface area.
»	Ends the defining procedure.
»	
ENTER ' TORSV STO▸	Puts the program on the stack. Stores the program in *TORSV*.

Now use *TORSV* to calculate the volume of a torus of inner radius $a = 6$ and outer radius $b = 8$.

6 ENTER 8 VAR TORSV

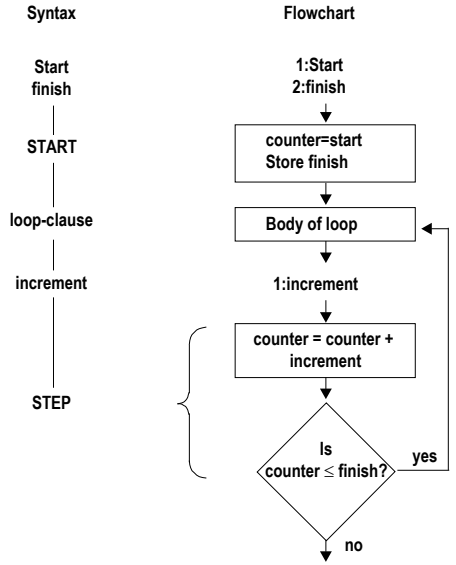
```
1:              138.174461616
 V2  V1  TST TORSV TORSA SPHLV
```


The START … STEP Structure

The syntax for this structure is

« … *start finish* START *loop-clause increment* STEP … »

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



START … STEP Structure

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

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

To enter START … STEP in a program:

■ Press ◀ PRG BRCH ▶ START.

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

« DUP → x « x 1 START x SQ -3 STEP » »

Example: System Flag. The following program sets an alarm for June 6, 2007 at 5:05 PM. It first tests the status of system flag –42 (Data Format flag) in a conditional structure and then supplies the alarm date in the current date format, based on the test result.

Example:

Program:	Comments:
«	
IF -42 FC?	Tests the status of flag –42, the Date Format flag.
THEN 6.152007	If flag –42 is clear, supplies the date in *month/day/year* format.
ELSE 15.062007	If flag –42 is set, supplies the date in *day.month.year* format.
END	Ends the conditional.
17.05 "TEST COMPLETE" 3 →LIST STOALARM	Sets the alarm: 17.05 is the alarm time and "TEST COMPLETE" is the alarm message.
»	

Example: User Flag. The following program returns either the fractional or integer part of the number in level 1, depending on the state of user flag 10.

Program:	Comments:
«	
IF	Starts the conditional.
10 FS?	Tests the status of user flag 10.
THEN IP	If flag 10 is set, returns the integer part.
ELSE FP	If flag 10 is clear, returns the fractional part.
END	Ends the conditional.
»	

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

Recalling and Storing the Flag States

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

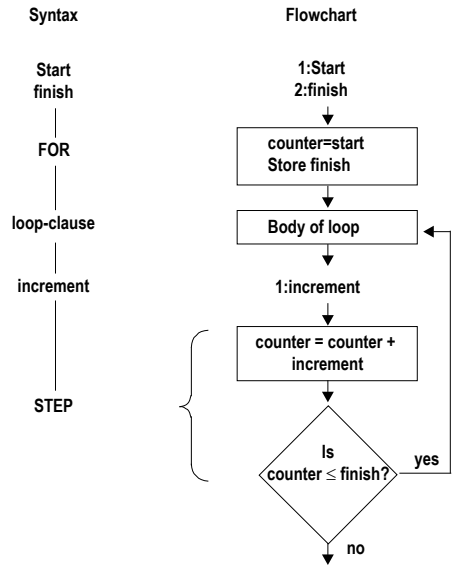
The RCLF (recall flags) and STOF (store flags) commands let you recall and store the states of the calculator's flags. For these commands, a 64-bit binary integer represents the states of 64 flags — each 0 bit corresponds to a flag that's clear, each 1 bit corresponds to a flag that's set. The rightmost (least significant) bit corresponds to system flag –1 or user flag 1 for the lower groups and system flag –65 or user flag 65 for the upper groups.

The FOR … STEP Structure

The syntax for this structure is

« … *start finish* FOR *counter loop-clause increment* STEP … »

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



FOR … STEP Structure

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

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

To enter FOR … STEP in a program:

■ Press ◀ PRG ▶ .

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

« 1 9 FOR × × SQ 2 STEP »

Using Summations Instead of Loops

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

Example: The following programs take an integer upper limit *n* from the stack, then find the summation. One program uses a FOR … NEXT loop — the other uses Σ.

$$\sum_{j=1}^{n} j^2$$

Program:	Comments:
≪	
0 1 ROT	Initializes the summation and puts the limits in place.
FOR j	Loops through the calculation.
j SQ +	
NEXT	
≫	

Program:	Comments:
≪	
→ n	Uses Σ to calculate the summation.
'Σ(j=1,n,j^2)'	
≫	

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

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

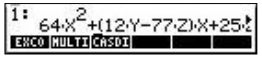
Example: The following program calculates $n + 2n + 3n + \ldots$ for a value of *n*. The program stops when the sum exceeds 1000, and returns the sum and the coefficient of *n*.

Program:	Comments:
≪	
DUP 1	Duplicates *n*, stores the value into *n* and *s*, and initializes *c* to *1*.
→ n s c	
≪	Starts the defining procedure.
DO	Starts the loop-clause.
'c' INCR	Increments the counter by 1. (See Using Loop Counters.)
n * 's' STO+	Calculates $c \times n$ and adds the product to *s*.
UNTIL	Starts the test-clause.
s 1000 >	Repeats loop until $s > 1000$.
END	Ends the test-clause.
s c	Puts *s* and *c* on the stack.
≫	Ends the defining procedure.
≫	

‚ 4 * Y + Z ' +

‚ 8 * X − 5 * Z

' y^x 2
ENTER



Select the VAR menu and start the program.

VAR EXCO

Minimum and Maximum Array Elements

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

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

MNX (Minimum or Maximum Element—Version 1)

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

Level 1	→	Level2	Level 1
[[*array*]]	→	[[*array*]]	Z_{min} *or* Z_{max}

Techniques used in MNX

- **DO…UNTIL…END (indefinite loop).** The DO clause contains the sort instructions. The UNTIL clause contains the system-flag test that determines whether to repeat the sort instructions.
- **User and system flags for logic control:**
  - *User* flag 10 defines the sort: When flag 10 is set, *MNX* finds the maximum element; when flag 10 is clear, it finds the minimum element. *You* determine the state of flag 10 at the beginning of the program.
  - *System* flag –64, the Index Wrap Indicator flag, determines when to end the sort. While flag –64 is clear, the sort loop continues. When the index invoked by GETI wraps back to the first array element, flag –64 is *automatically* set, and the sort loop ends.
- **Nested conditional.** An IF…THEN…END conditional is nested in the DO…UNTIL…END conditional, and determines the following:
  - Whether to maintain the current minimum or maximum element, or make the current element the new minimum or maximum.
  - The sense of the comparison of elements (either < or >) based on the status of flag 10.
- **Custom menu.** *MNX* builds a custom menu that lets you choose whether to sort for the minimum or maximum element. Key 1, labeled MAX, sets flag 10. Key 2, labeled MIN, clears flag 10.
- **Logical function.** *MNX* executes XOR (*exclusive OR*) to test the combined state of the relative value of the two elements and the status of flag 10.

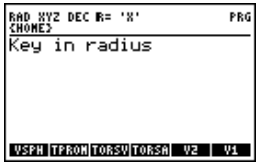
The following program, *VSPH*, calculates the volume of a sphere. *VSPH* prompts for the radius of the sphere, then cubes it and multiplies by $^4/_3\ \pi$. *VSPH* executes INPUT to prompt for the radius. INPUT sets Program-entry mode when program execution pauses for data entry.

Program:	Comments:
«	
"Key in radius"	Specifies the prompt string.
""	Specifies the command-line string. In this case, the command line will be empty.
INPUT	Displays the prompt, puts the cursor at the start of the command line, and suspends the program for data input (the radius of the sphere).
OBJ→	Converts the result string into its component object — a real number.
3 ^	Cubes the radius.
4 * 3 / π * →NUM	Completes the calculation.
»	
ENTER ' VSPH STO	Stores the program in VSPH.

Example:

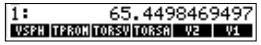
Execute VSPH to calculate the volume of a sphere of radius 2.5.

VAR VSPH



Key in the radius and continue program execution.

2.5 ENTER



MULTI (Multiple Execution)

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

Level 2	Level 1	→	Level 1
object	*«program»*	→	*object*$_{result}$

Techniques used in MULTI

- **DO…UNTIL…END (indefinite loop).** The DO clause contains the steps to be repeated. The UNTIL clause contains the test that repeats both clauses again (if false) or exits (if true).
- **Programs as arguments.** Although programs are commonly named and then executed by calling their names, programs can also be put on the stack and used as arguments to other programs.
- **Evaluation of local variables.** The program argument to be executed repeatedly is stored in a local variable.

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

MULTI program listing

Program:	Comments:
«	
→ p	Creates a local variable *p* that contains the program from level 1.
«	Begins the defining procedure.
DO	Begins the DO loop clause.
DUP	Makes a copy of the object, now in level 1.
p EVAL	Applies the program to the object, returning its new version.
DUP	Makes a copy of the new object.
ROT	Moves the old version to level 1.
UNTIL	Begins the DO test clause.
SAME	Tests whether the old version and the new version are the same.
END	Ends the DO structure.
»	Ends the defining procedure.
»	
ENTER ' MULTI STO▸	Stores the program in *MULTI*.

Checksum: # 22693d
Bytes: 56

MULTI is demonstrated in the next programming example.

To process the result string from INPUT:

- For simple input, use OBJ→ to convert the string into its corresponding objects.
- For sensitive input, use the V option for INPUT to check for valid objects, then use OBJ→ to convert the string into those objects.
- For special input, process the input as a string object, possibly extracting data as substrings.

Example: The program *VSPH* on page 1-41 uses an empty command-line string.

The program *SSEC* on page 1-44 uses a command-line string whose characters form a pattern. The program extracts substrings from the result string.

Example: The command-line string `"@UPPER LIMIT@"` displays `@UPPER LIMIT@` in the command line. If you press 200 ENTER the return string is `"@UPPER LIMIT@200"`. When OBJ→ extracts the text from the string, it strips away the @ characters and the enclosed characters, and it returns the number 200. (See "Creating Programs on a computer" on page 1-7 for more information about @ comments.)

Example: The following program, TINPUT, executes INPUT to prompt for the inner and outer radii of a torus, then calls TORSA (page 1-29) to calculate its surface area. *TINPUT* prompts for *a* and *b* in a two-row command line. The level 1 argument for INPUT is a list that contains:

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

Program:	Comments:
«	
"Key in a, b"	The level 2 string, displayed at the top of the stack area.
{ ":a:↵:b:" {1 0} V }	The level 1 list contains a string, a list, and the verify option. (To key in the string, press ⇧ " ⇧ : a ▶ ⇧ ⏎ ⇧ : b. After you press ENTER to put the finished program on the stack, the string is shown on one line, with ↵ indicating the newline character.) The embedded list puts the insert cursor at the end of row 1.
INPUT	Displays the stack and command-line strings, positions the cursor, sets Program-entry mode, and suspends execution for input.
OBJ→	Converts the string into its component objects — two tagged objects.
TORSA	Calls *TORSA* to calculate the surface area.
»	
ENTER ' TINPUT STO▸	Stores the program in *TINPUT*.

Techniques used in MEDIAN

■ **Arrays, lists, and stack elements.** *MEDIAN* extracts a column of data from *ΣDAT* in vector form. To convert the vector to a list, *MEDIAN* puts the vector elements on the stack and combines them into a list. From this list the median is calculated using *%TILE*.

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

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

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

Required Program

■ *%TILE* (page 2-10) sorts a list and returns the value of a specified percentile.

MEDIAN program listing (Note: Use approximate mode for this program and example).

Program:	Comments:
«	
RCLΣ	Puts a copy of the current statistics matrix *ΣDAT* on the stack.
DUP SIZE	Puts the list { *n m* } on the stack, where *n* is the number of rows in *ΣDAT* and *m* is the number of columns.
OBJ→ DROP	Puts *n* and *m* on the stack, and drops the list size.
→ s n m	Creates local variables for *s*, *n*, and *m*.
«	Begins the defining procedure.
'ΣDAT' TRN	Recalls and transposes *ΣDAT*. Now *n* is the number of columns in *ΣDAT* and *m* is the number of rows. (To key in the Σ character, press ⌐ —Σ, then delete the parentheses.)
1 m	Specifies the first and last rows.
FOR j	For each row, does the following: Extracts the last row in *ΣDAT*. Initially this is the *m*th row, which corresponds to the *m*th column in the original *ΣDAT*. (To key in the Σ– command, use ⌐ —CAT .)
Σ–	
OBJ→ DROP	Puts the row elements on the stack. Drops the index list { *n* }.
n →LIST	Makes an *n*-element list.
50 %TILE	Sorts the list and calculates its median.
j ROLLD	Moves the median to the proper stack level.
NEXT	Increments *j* and repeats the loop.

Program:	Comments:
«	
"Key in S.S. #"	Prompt string.
{ " – – " -1 }	Command-line string (3 spaces before the first –, 2 spaces between, and 4 spaces after the last –).
INPUT	Suspends the program for input.
DUP 1 3 SUB SWAP 8 11 SUB	Copies the result string, then extracts the first three and last four digits in string form.
»	
' SSEC STO▸	Stores the program in *SSEC*.

Using INFORM and CHOOSE for Input

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

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

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

To set up an input form:

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

Example: Enter a title "FIRST ONE" ENTER.
Specify a field { "Name:" } ENTER.
Enter format options (one column, tabs stop width five) { 1 5 } ENTER.

Enter reset value for the field { "THERESA" } ENTER.
Enter default value for the field { "WENDY" } ENTER.
Execute INFORM (↰ PRG NXT INFORM).
The screen on the left appears. Press NXT RESET OK and the screen on the right appears.

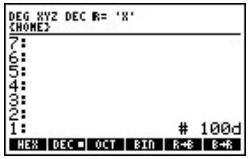
Example: Switch to DEC base, display #100 in all bases, and check that BDISP restored the base to DEC.

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

CLEAR

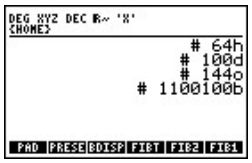
BASE DEC

100 ENTER



Execute *BDISP*.

VAR BDISP

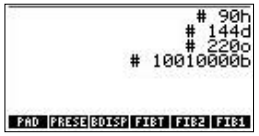


Return to the normal stack display and check the current base.

CANCEL

BASE

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



VAR

144 BDISP

Press CANCEL to return to the stack display.

Median of Statistics Data

This section contains two programs:

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

%TILE (Percentile of a list)

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

Level 2	Level 1	→	Level 1
{ list }	*n*	→	*n*th *percentile of sorted list*

Program:	Comments:
`"ADD A NAME"`	Creates an input form that gets the name and phone number. The two fields accept only strings (object type 2).
`{ { "NAME:" "ENTER NAME" 2 } { "PHONE:" "ENTER A PHONE NUMBER" 2 } } { } { } { } INFORM REPEAT`	
`DUP IF { NOVAL } HEAD POS THEN`	Checks if either field in the new entry is blank.
`DROP "Complete both fields before pressing OK" MSGBOX`	If either one is, displays a message.
`ELSE 1 →LIST NAMES + SORT 'NAMES' STO`	If neither are, adds the list to NAMES, sorts it, and stores it back in NAMES.
`END END END`	Ends the IF structure, the WHILE loop, and the CASE statement.
`c 2 ==`	Case 2 (View a Number).
`THEN IF { } NAMES SAME THEN`	Checks if NAMES is an empty list.
`"YOU MUST ADD A NAME FIRST" MSGBOX`	If it is, displays a message.
`ELSE WHILE "VIEW A NUMBER" NAMES 1 CHOOSE REPEAT →STR MSGBOX`	If NAMES isn't empty, creates a choose box using NAMES as choice items. When OK is pressed, the second item in the NAMES list pairs (the phone number) is returned. Makes it a string and displays it.
`END END END`	Ends the WHILE loop, the IF structure, and the CASE statement.
`END » END »`	Ends the CASE structure, marks the end of the local variable defining procedure, ends the WHILE loop, and marks the end the program.
ENTER ' PHONES STO	Stores the program in *PHONES*.

Techniques used in BDISP

- **IFERR…THEN…END (error trap).** To accommodate real-number arguments, *BDISP* includes the command R→B (*real-to-binary*). However, this command causes an error if the argument is *already* a binary integer. To maintain execution if an error occurs, the R→B command is placed inside an IFERR clause. No action is required when an error occurs (since a binary number is an acceptable argument), so the THEN clause contains no commands.
- **Enabling LASTARG.** In case an error occurs, the LASTARG recovery feature must be enabled to return the argument (the binary number) to the stack. *BDISP* clears flag –55 to enable this.
- **FOR…NEXT loop (definite loop with counter).** *BDISP* executes a loop from 1 to 4, each time displaying *n* (the number) in a different base on a different line. The loop counter (named *j* in this program) is a local variable created by the FOR…NEXT program structure (rather than by a → command), and automatically incremented by NEXT.
- **Unnamed programs as arguments.** A program defined only by its « and » delimiters (not stored in a variable) is not automatically evaluated, but is placed on the stack and can be used as an argument for a subroutine. *BDISP* demonstrates two uses for unnamed program arguments:
  - *BDISP* contains a main program argument and a call to *PRESERVE*. This program argument goes on the stack and is executed by *PRESERVE*.
  - *BDISP* also contains four program arguments that "customize" the action of the loop. Each of these contains a command to change the binary base, and each iteration of the loop evaluates one of these arguments.

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

Required Programs

PAD

- PAD (Pad with Leading Spaces) expands a string to 22 characters so that DISP shows it right-justified.

PRESERVE

- PRESERVE stores the current status, executes the main nested program, and restores the status.

Using KEY for Keystroke Input

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

To enter a KEY loop in a program

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

KEY returns 0 to level 1 when the loop begins. It continues to return 0 until a key is pressed — then it returns 1 to level 1 and the two-digit row-column number of the pressed key to level 2. For example, ENTER returns 105, and ◀ returns 81.)

The test-clause should normally cause the loop to repeat until a key is pressed. If a key is pressed, you can use comparison tests to check the value of the key number. (See "Using Indefinite Loop Structures" on page 1-22 and "Using Comparison Functions" on page 1-11.)

To respond to a KEY loop while running a program:

- Press any key. (A prefix key such as ◀ or ALPHA is a valid key.)

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

```
« … DO UNTIL KEY END 95 == … »
```

Output

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

Data Output Commands

Key	Command	Description
◀ PRG NXT [illegible]:		
[illegible]	PVIEW	Displays PICT starting at the given coordinates.
[illegible]	TEXT	Displays the stack display.
[illegible]	CLLCD	Blanks the stack display.
[illegible]	DISP	Displays an object in the specified line.
[illegible]	FREEZE	"Freezes" a specified area of the display until a key press.
[illegible]	MSGBOX	Creates a user-defined message box.
[illegible]	BEEP	Sounds a beep at a specified frequency (in hertz, level 2) and duration (in seconds, level 1).

PAD program listing

Program:	Comments:
«	
→STR	Makes sure the object is in string form. (Strings are unaffected by this command.)
WHILE DUP SIZE 22 <	Repeats if the string contains fewer than 22 characters.
REPEAT " " SWAP +	Loop-clause adds a leading space.
END	End loop.
»	
ENTER ' PAD STO▸	Stores the program in *PAD*.

Checksum: # 6577d
Bytes: 57.5

PAD is demonstrated in the program *BDISP*.

PRESERVE (Save and Restore Previous Status)

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

Level 1	→	Level 1
«program»	→	*result of program*
'program'	→	*result of program*

Techniques used in PRESERVE

- **Preserving calculator flag status.** *PRESERVE* uses RCLF (*recall flags*) to record the current status of the calculator in a binary integer, and STOF (*store flags*) to restore the status from that binary integer.
- **Local-variable structure.** *PRESERVE* creates a local variable structure to briefly remove the binary integer from the stack. Its defining procedure simply evaluates the program argument, then puts the binary integer back on the stack and executes STOF.
- **Error trapping.** *PRESERVE* uses IFERR to trap faulty program execution on the stack and to restore flags. DOERR shows the error if one occurs.

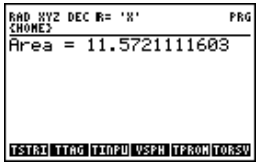
Example: The following program *TSTRING* is identical to *TINPUT*, except that it converts the program result to a string and appends a labeling string to it.

Program:	Comments:
«	
"Key in a, b"	Converts the result to a string.
{ ":a:↵:b:" (1 0) V} INPUT OBJ→	Enters the labeling strings.
TORSA →STR "Area = "	
SWAP +	Swaps and adds the two strings.
CLLCD 1 DISP 3 FREEZE	Displays the resultant string, without its delimiters, in line 1 of the display.
»	
ENTER ' TSTRING STO▸	Stores the program in *TSTRING*.

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

◱ CLEAR VAR TSTRI

1.5 ▼ 1.85 ENTER



Pausing to Display Output

To pause to display a result:

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

WAIT suspends execution for the number of seconds in level 1. You can use WAIT with DISP to display messages during program execution — for example, to display intermediate program results. (WAIT interprets arguments 0 and –1 differently — see "Using WAIT for Keystroke Input" on page 1-48.)

Using MSGBOX to Display Output

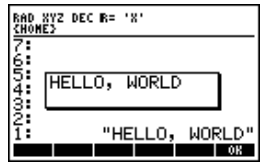
To set up a message box:

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

Example: Enter the string "HELLO, WORLD" ENTER.

Execute MSGBOX (◱ PRG NXT MSGBO).

The following message appears:



FIBT (Comparing Program-Execution Time)

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

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

Level 1	→	Level 3	Level 2	Level 1
n	→	F_n	FIB1 TIME: z	FIB2 TIME: z

Techniques used in FIBT

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

Required Programs

- *FIB1* (page 2-1) calculates F_n using recursion.
- *FIB2* (page 2-2) calculates F_n using looping.

FIBT program listing

Program:	Comments:
«	
DUP TICKS SWAP FIB1 SWAP TICKS SWAP	Copies *n*, then executes *FIB1*, recording the start and stop time.
- B→R 8192 /	Calculates the elapsed time, converts it to a real number, and converts that number to seconds. Leaves the answer returned by *FIB1* in level 2.
"FIB1 TIME" →TAG	Tags the execution time.
ROT TICKS SWAP FIB2 TICKS	Executes *FIB2*, recording the start and stop time.
SWAP DROP SWAP - B→R 8192 /	Drops the answer returned by *FIB2* (*FIB1* returned the same answer). Calculates the elapsed time for *FIB2* and converts to seconds.
"FIB2 TIME" →TAG	Tags the execution time.
»	
ENTER ' FIBT STO▸	Stores the program in *FIBT*.

Checksum: # 23164d
Bytes: 129

The program remains halted until it's resumed by a CONT command, such as by pressing ◀ CONT . If you create a custom menu for input, you can include a CONT command to automatically resume the program when you press the menu key.

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

```
« 65 MENU "Select Angle Mode" PROMPT »
```

Example: The *PIE* program on page 2-34 assigns the CONT command to one key in a temporary menu.

Example: The *MNX* program on page 2-16 sets up a temporary menu that includes a program containing CONT to resume execution automatically.

Using Menus to Run Programs

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

To create a menu-based application:

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

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

```
{
 { "ENGL" « "ENTER Wt in POUNDS" PROMPT 32.2 / » }
 { "SI" « "ENTER Wt in NEWTONS" PROMPT 9.81 / » }
}
```

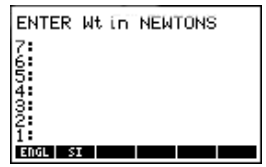
' LST STO▸

Program:	Comments:
« LST TMENU »	Displays the custom menu stored in *LST*.
ENTER ' WGT STO▸	Stores the program in *WGT*.

Use *WGT* to calculate the mass of an object of weight 1.25 N. The program sets up the menu, then completes execution.

VAR

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



Key in the weight, then resume the program.

Techniques used in FIB1

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

FIB1 program listing

Program:	Comments:
« → n 'IFTE(n≤1, n, FIB1(n-1)+FIB1(n-2))' »	Defines local variable *n*. The defining procedure, an algebraic expression. If $n \leq 1$, $F_n = n$, else $F_n = F_{n-1}+F_{n-2}$.
ENTER ' FIB1 STO▶	Stores the program in *FIB1*.

Checksum: # 14909d (press ' ▒▒▒ ◀ PRG ▒▒▒ ▒▒▒)
Bytes: 113.5

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

First calculate F_6.

VAR

6 ▒▒▒

Next, calculate F_{10} using algebraic syntax.

' ▒▒▒ ◀ ( ) 10 EVAL

FIB2 (Fibonacci Numbers, Loop Version

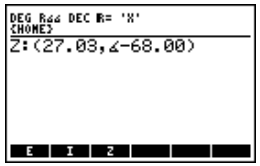
Level 1	→	Level 1
n	→	F_n

Techniques used in FIB2

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

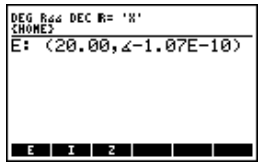
Store the voltage value. Then key in and store the current value. Solve for the impedance.

▒▒▒ ◀ ( ) .37 ALPHA ▶ 6 68 ▒▒▒
◀ ▒▒▒



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

▶ ▒▒▒ 2 × ▒▒▒ ◀ ▒▒▒



Press (◀ &MODE) ▒▒▒ ▒▒▒ and NXT ▒▒▒ ▒▒▒ ▒▒▒ to restore Standard and Rectangular modes.

Turning Off the Calculator from a Program

To turn off the calculator in a program:

- Execute the OFF command (PRG RUN menu).

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

$$arc\cos\left(\frac{2}{3}\right) + arc\cos(x)$$

Command: `ACOS2S(ACOS(2/3)+ACOS(X))`

Result: `π/2-ASIN(2/3)+π/2-ASIN(X)`

See also: ASIN2C, ASIN2T, ATAN2S

ACOSH

Type: Analytic Function

Description: Inverse Hyperbolic Cosine Analytic Function: Returns the inverse hyperbolic cosine of the argument.

For real arguments $x < 1$, ACOSH returns the complex result obtained for the argument $(x, 0)$.

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

`s1*ACOSH(Z)+2*π*i*n1`

The function ACOSH is the inverse of a *part* of COSH, a part defined by restricting the domain of COSH such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

The points in this restricted domain of COSH are called the *principal values* of the inverse relation. ACOSH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ACOSH to the boundary of the restricted domain of COSH form the *branch cuts* of ACOSH.

The principal branch used by the calculator for ACOSH was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued hyperbolic arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ACOSH. The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation s1*ACOSH(Z)+2*π*i*n1 for the case $s1 = 1$ and $n1 = 0$. For other values of $s1$ and $n1$, the horizontal half-band in the lower graph is rotated to the left and translated up and down. Taken together, the bands cover the whole complex plane, which is the domain of COSH.

View these graphs with domain and range reversed to see how the domain of COSH is restricted to make an inverse *function* possible. Consider the horizontal half-band in the lower graph as the restricted domain $Z = (x, y)$. COSH sends this domain onto the whole complex plane in the range $W = (u, v) = \text{COSH}(x, y)$ in the upper graph.

Access: [→] [TRIG] HYPERBOLIC ACOSH ([TRIG] is the right-shift of the [8] key).

Flags: Principal Solution (–1), Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*acosh* z
'symb'	→	*'ACOSH(symb)'*

See also: ASINH, ATANH, COSH, ISOL

MNX program listing

Program:	Comments:
`«`	
`<< "MAX" « 10 SF CONT » 'MIN" « 10 CF CONT » >>`	Defines the option menu. [MAX] sets flag 10 and continues execution. [MIN] clears flag 10 and continues execution.
`TMENU "Sort for MAX or MIN?" PROMPT`	Displays the temporary menu and a prompt message.
`1 GETI`	Gets the first element of the array.
`DO`	Begins the DO loop.
`ROT ROT GETI`	Puts the index and the array in levels 1 and 2, then gets the new array element.
`4 ROLL DUP2`	Moves the current minimum or maximum array element from level 4 below 1, then copies both.
`IF > 10 FS? XOR`	Tests the combined state of the relative value of the two elements and the status of flag 10.
`THEN SWAP END`	If the new element is either less than the current maximum or greater than the current minimum, swaps the new element into level 1.
`DROP`	Drops the other element off the stack.
`UNTIL -64 FS?`	Begins the DO test-clause. Tests if flag –64 is set — if the index reached the end of the array.
`END`	Ends the DO loop.
`SWAP DROP 0 MENU`	Swaps the index to level 1 and drops it. Restores the last menu.
`»`	
[ENTER] ['] MNX [STO▸]	Stores the program in *MNX*.

Checksum: # 20991d
Bytes: 194.5

Example: Find the maximum element of the following matrix:

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

ACK has no effect on control alarms. Control alarms that come due are automatically acknowledged *and* saved in the system alarm list.

Access: ▶ TIME TOOLS ALRM ACK (TIME is the right-shift of the 9 key).

Flags: Repeat Alarms Not Rescheduled (–43), Acknowledged Alarms Saved (–44)

Input/Output: None

See also: ACKALL

ACKALL

Type: Command

Description: Acknowledge All Alarms Command: Acknowledges all past-due alarms.

ACKALL clears the alert annunciator if there are no other active alert sources (such as a low battery condition).

ACKALL has no effect on control alarms. Control alarms that come due are automatically acknowledged *and* saved in the system alarm list.

Access: ▶ TIME TOOLS ALRM ACKALL (TIME is the right-shift of the 9 key).

Flags: Repeat Alarms Not Rescheduled (–43), Acknowledged Alarms Saved (–44)

Input/Output: None

See also: ACK

ACOS

Type: Analytic Function

Description: Arc Cosine Analytic Function: Returns the value of the angle having the given cosine.

For a real argument x in the domain $-1 \le x \le 1$, the result ranges from 0 to 180 degrees (0 to π radians; 0 to 200 grads).

A real argument outside of this domain is converted to a complex argument, $z = x + 0i$, and the result is complex.

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

```
s1*ACOS(Z)+2*π*n1
```

The function ACOS is the inverse of a *part* of COS, a part defined by restricting the domain of COS such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

The points in this restricted domain of COS are called the *principal values* of the inverse relation. ACOS in its entirety is called the *principal branch* of the inverse relation, and the points sent by ACOS to the boundary of the restricted domain of COS form the *branch cuts* of ACOS.

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

The graphs below show the domain and range of ACOS. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation s1*ACOS(Z)+2*π*n1 for the case $s1=1$ and $n1 = 0$. For other values of *s1* and *n1*, the vertical band in the lower graph is translated to the right or to the left. Taken together, the bands cover the whole complex plane, which is the domain of COS.

MNX2 program listing

Program:	Comments:
«	
<< "MAX" « 10 SF CONT » "MIN" « 10 CF CONT » >>	Defines the temporary option menu. MAX sets flag 10 and continues execution. MIN clears flag 10 and continues execution.
TMENU	Displays the temporary menu and a
"Sort for MAX or MIN?" PROMPT	prompting message.
DUP OBJ→	Copies the array. Returns the individual array elements to levels 2 through *nm*+1, and returns the list containing *n* and *m* to level 1.
1	Sets the initial counter value.
SWAP OBJ→	Converts the list to individual elements on the stack.
DROP * 1 –	Drops the list size, then calculates the final counter value (*nm* - 1).
FOR n	Starts the FOR…NEXT loop.
DUP2	Saves the array elements to be tested (initially the last two elements). Uses the last array element as the current minimum or maximum.
IF > 10 FS? XOR	Tests the combined state of the relative value of the two elements and the status of flag 10.
THEN SWAP END	If the new element is either less than the current maximum or greater than the current minimum, swaps the new element into level 1.
DROP	Drops the other element off the stack.
NEXT	Ends the FOR…NEXT loop.
0 MENU	Restores the last menu.
»	
ENTER ' MNX2 STO▸	Stores the program in *MNX2*.

Checksum: # 6992d
Bytes: 188.5

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

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

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

Terms Used in Stack Diagrams

Term	Description
arg	Argument.
[*array*]	Real or complex vector or matrix.
[*C-array*]	Complex vector or matrix.
date	Date in form MM.DDYYYY or DD.MMYYYY.
{ *dim* }	List of one or two array dimensions (real numbers).
'global'	Global name.
grob	Graphics object.
HMS	A real-number time or angle in hours-minutes-seconds format.
{ *list* }	List of objects.
local	Local name.
[[*matrix*]]	Real or complex matrix.
n or m	Positive integer real number (rounded if noninteger)
:n_{port}:	Backup identifier.
:n_{port}: $n_{library}$	Library identifier.
#n	Binary integer.
{ *#n #m* }	Pixel coordinates. (Uses binary integers.)
'name'	Global or local name.
obj	Any object.
PICT	Current graphics object.
« program »	Program.
[*R-array*]	Real vector or matrix.
"string"	Character string.
'symb'	Expression, equation, or name treated as an algebraic.
T/F	Test result used as an argument: zero (false) or non-zero (true) real number.
0/1	Test result *returned* by a command: zero (false) or one (true).
time	Time in form HH.MMSSs.
[*vector*]	Real or complex vector.
x or y	Real number.
x_unit	Unit object, or a real number treated as a dimensionless object.
(x,y)	Complex number in rectangular form, or user-unit coordinate.
z	Real or complex number.

Program:	Comments:
`1 CF`	Make sure the flag 1 is clear to begin the procedure.
`a DUP SIZE`	Retrieve the dimensions of the array.
`DUP SIZE`	Determine if the array is a vector.
`IF 1 ==`	
`THEN 1 SF 1 +`	If array is a vector, set flag 1 and add a second dimension by treating the vector as an $n \times 1$ matrix.
`SWAP OBJ→ OBJ→ DROP`	Disassemble the original vector, leaving the element count, *n*, in level 1.
`1 + ROLL`	Roll the elements up the stack and bring the "matrix" dimensions of the vector to level 1.
`ELSE DROP2 a OBJ→`	If array is a matrix, clean up the stack and decompose the matrix into its elements, leaving its dimension list on level 1.
`END DUP OBJ→ DROP *`	Duplicate the dimension list and compute the total number of elements.
`SWAP OVER 2 +` `ROLLD →LIST`	Roll up the element count and combine all elements into a list. Note that the elements in the list are in row-major order.
`1 p DOSUBS`	Recalls the program and uses it as an argument for DOSUBS (DOLIST works in this case as well). Result is a list of transformed elements.
`OBJ→ 1 + ROLL`	Disassembles the result list and brings the array dimensions to level 1.
`IFERR`	Begins the error-trapping structure. Its purpose is to find and handle the cases when the result list contains symbolic elements.
`IF 1 FS?` `THEN OBJ→ DROP` `→LIST`	Was original array a vector? If the original array was a vector, then drop the second dimension (1) from the dimension list.
`END →ARRY`	Convert the elements into an array with the given dimensions. If there are symbolic elements present, an error will be generated and the error clause which follows will be executed.
`THEN`	Begin the error clause.
`OBJ→`	Put the array dimensions on levels 2 and 1. If the array is a vector, level 1 contains a 1.
`IF 1 FC?C`	Is original array a matrix? Clear flag 1 after performing the test.
`THEN DROP`	Drop the number of matrix elements.
`END → n m`	Store the array dimensions in local variables.
`« 1 n`	Begin local variable structure and initiate
`FOR i`	FOR…NEXT loop for each row.
`m →LIST`	Collect a group of elements into a row (a list).
`'m*(n-i)+i' EVAL`	Computes the number of elements to roll so
`ROLLD`	that the next row can be collected.

first case, it returns the numeric arccosine; in the second, it returns the symbolic arccosine expression of the argument.

Some commands affect a calculator state — a mode, a reserved variable, a flag, or a display — without taking any arguments from the stack or returning any results to the stack. No stack diagrams are shown for these commands.

Other commands may have more complicated input or output that is easier to explain in prose. These commands do not show stack diagrams either and instead have separate **Input** and **Output** sections.

Other Provided Details

In addition to the **Input/Output** and **Type**, for each operation in the alphabetical list, some or all of the following details are provided:

Description: A description of the operation.

Access: The menu or choose-list on which an operation can be found, and the keys that you press to access it. If the operation is on a sub-menu, the sub-menu name is in SMALL CAPITALS after the keys. CAS commands that are not in any of the other menus on the keyboard can be accessed from the ↱ CAT menu. Most CAS commands can also be accessed from the CASCMD choose-list, from CAS soft menus and from menus created by the MENUXY command.

Flags: Details of which flag settings affect the operation of the function or command. See also the section below on CAS Settings.

Example: An example of the function or command. Some examples are also available in the built-in CAS help on the calculator or in chapters 11 to 16 in the User's Guide. Most of the examples given here are shown in Algebraic mode, but can be transferred to RPN mode according to the descriptions given in "Input" and "Output".

See also: Related functions or commands.

Parallel Processing with Lists

This feature is discussed in greater detail in Appendix G.

As a rule-of-thumb, a command can use parallel list processing if all the following are true:

- The command checks for valid argument types. Commands that apply to all object types, such as DUP, SWAP, ROT, and so forth, do not use parallel list processing.
- The command takes exactly one, two, three, four, or five arguments, none of which may itself be a list. Commands, such as →LIST, that have an indefinite number of arguments do not use parallel list processing.
- The command is not a programming branch command (IF, FOR, CASE, NEXT, and so forth).

There are also a few commands (PURGE, DELKEYS, SF and FS? are examples) that have list processing capability built into their definitions, and so do not also use the parallel list processing feature.

How Commands Are Alphabetized

Commands appear in alphabetical order. Command names that contain special (non-alphabetic) characters are organized as follows:

- For commands that contain *both* special and alphabetic characters:
  - A special character at the *start* of a command name is *ignored*. Therefore, the command %CH follows the command CF and precedes the command CHOOSE.
  - A special character *within* or at the *end* of a command name is considered to follow "Z" at the end of the alphabet. Therefore, the command R→B follows the command RSWP and precedes the command R→C. The only exception would be the "Σ" character which, when not the first character in the name, is alphabetized as if it were the string "SIGMA". An example is "NΣ", which falls between NOVAL and NSUB.
- Commands that contain *only* special characters appear at the end of the dictionary.

nBASE program listing

Program:	Comments:
«	
1 CF 0 RND SWAP 0 RND RCLF	Clear flag 1, round both arguments to integers and recall flag settings.
→ b n f	Store the base, number and flag settings in local variables.
«	Begin the outer local variable structure.
STD n LOG b LOG /	Sets "standard" display mode and computes the ratio of the common logarithms of number and base.
10 RND	Rounds result to remove imprecision in last decimal place.
IP n 0	Find the integer part of log ratio, recall the original number, and initialize the counter variable *k* for use in the DO...UNTIL loop.
→ i m k	Store the values in local variables.
«	Begin inner local variable structure,
""	enter an empty string and begin the
DO	DO...UNTIL...END loop.
'm' EVAL b i	Compute the decimal value of the
'k' EVAL - ^	(*i* – *k*) th position in the string.
DUP2 MOD	Makes a copy of the arguments and computes the decimal value still remaining that must be accounted for by other positions.
IF DUP 0 ==	Is the remainder zero *and* $m \geq b$?
'm' EVAL b ≥	
AND	
THEN 1 SF	If the test is true, then set flag 1.
END 'm' STO	Store the remainder in *m*. Compute the
/ IP	number of times the current position-value goes into the remaining decimal value. This is the "digit" that belongs in the current position.
IF DUP 10 ≥	Is the "digit" ≥ 10?
THEN 55	Then convert the digit into a alphabetic
ELSE 48	digit (such as A, B, C, …).
END + CHR	
+ 'k' 1 STO+	Append the digit to the current result string and increment the counter variable *k*.

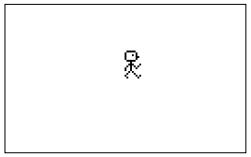
Program:	Comments:
`« # 0d # 25d`	Puts the first position on the stack and turns on the first image. This readies the stack and *PICT* for the loop.
`PICT OVER walk GXOR`	
`5 MAXR FOR i`	Starts the loop to generate horizontal coordinates indefinitely.
`i 131 MOD R→B`	Computes the horizontal coordinate for the next image.
`# 25d 2 →LIST`	Specifies a fixed vertical coordinate. Puts the two coordinates in a list.
`PICT OVER walk GXOR`	Displays the new image, leaving its coordinates on the stack.
`PICT ROT walk GXOR`	Turns off the old image, removing its coordinates from the stack.
`0.2 WAIT 5 STEP`	Increments the horizontal coordinate by 5.
`»`	
`»`	
ENTER ' WALK STO	Stores the program in *WALK*.

Checksum: # 28684d
Bytes: 250.0

Example: Send the small person out for a walk.

VAR

Press CANCEL when you think the walker's tired.



NAMES (Check List for Exactly Two Names)

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

Level 1	→	Level 1
{ valid list }	→	
{ invalid list }	→	*(error message in status area)*

Techniques used in NAMES

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

NAMES program listing

Program:	Comments:
`«`	
`IF`	Starts the outer conditional structure.
`OBJ→`	Returns the *n* objects in the list to levels 2 through (*n* + 1), and returns the list size *n* to level 1.
`DUP 2. SAME`	Copies the list size and tests if it's 2.
`THEN`	If the size is 2, moves the objects to level 1 and 2, and starts the inner conditional structure.
`DROP`	
`IF`	
`TYPE 6. SAME`	Tests if the object is a name: returns 1 if so, otherwise 0.
`SWAP TYPE 6. SAME`	Moves the second object to level 1, then tests if it is a name (returns 1 or 0).
`AND`	Combines test results: Returns 1 if both tests were true, otherwise returns 0.
`NOT`	Reverses the final test result.
`THEN`	If the objects are not both names, displays an error message and aborts execution.
`"List needs two names"`	
`DOERR`	
`END`	Ends the inner conditional structure.
`ELSE`	If the list size is not 2, drops the list size, displays an error message, and aborts execution.
`DROPN`	
`"Illegal list size"`	
`DOERR`	
`END`	Ends the outer conditional.
`»`	
ENTER ' NAMES STO	Stores the program in *NAMES*.

Inverse-Function Solver

This section describes the program *ROOTR*, which finds the value of *x* at which *f(x)* = *y*. You supply the variable name for the program that calculates *f(x)*, the value of *y*, and a guess for *x* (in case there are multiple solutions).

Level 3	Level 2	Level 1	→	Level 1
'function name'	*y*	x_{guess}	→	*x*

Techniques used in ROOTR

- **Programmatic use of root-finder.** *ROOTR* executes ROOT to find the desired *x*-value.
- **Programs as arguments.** Although programs are commonly named and then executed by calling their names, programs can also be put on the stack and used as arguments to other programs.

ROOTR program listing

Program:	Comments:
`« → fname yvalue xguess « xguess 'XTEMP' STO « XTEMP fname yvalue - » 'XTEMP' xguess ROOT » 'XTEMP' PURGE »`	Creates local variables. Begins the defining procedure. Creates variable *XTEMP* (to be solved for). Enters program that evaluates *f(x)* - *y*. Enters name of unknown variable. Enters guess for *XTEMP*. Solves program for *XTEMP*. Ends the defining procedure. Purges the temporary variable.
ENTER ' ROOTR STO	Stores the program in *ROOTR*.

Checksum: # 4708d
Bytes: 163

Example: Assume you often work with the expression $3.7x^3 + 4.5x^2 + 3.9x + 5$ and have created the program *X→FX* to calculate the value:

```
« → x '3.7*x^3+4.5*x^2+3.9*x+5' »
```

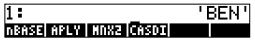
You can use *ROOTR* to calculate the *inverse* function.

Program:	Comments:
`« argm TYPE 6. SAME NOT THEN "Not name or list" DOERR END END »`	Tests if the argument is not a name. If so, displays an error message and aborts execution. Ends the CASE structure. Ends the defining procedure.
ENTER ' VFY STO	Enters the program, then stores it in *VFY*.

Checksum: # 31403d
Bytes: 139.5

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

' BEN ENTER



VAR VFY

Example: Execute *VFY* to test the validity of the list argument {*BEN JEFF SARAH* }. Use the name from the previous example, then enter the names JEFF and *SARAH* and convert the three names to a list.

' JEFF ENTER

' SARAH ENTER

3 ← PRG LIST →LIST

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

VAR VFY

Converting Procedures from Algebraic to RPN

This section contains a program, →*RPN*, that converts an algebraic expression into a series (list) of objects in equivalent RPN order.

Level 1	→	Level 1
'symb'	→	{ *objects* }

Program:	Comments:

`prcnts n GET`
`1 RND`
`→STR "%" +`
	Gets the *n*th value from the percentage matrix, rounds it to one decimal place, and converts it to a string with "%" appended.

`1 →GROB`	Converts the string to a graphics object.
`GOR DUP PICT STO`	Adds the label to the plot and stores the new plot.

`→LCD`
`NEXT`
`{ } PVIEW`
	Displays the updated plot.
	Ends the loop structure.
	Displays the finished plot.

`»`
`»`
`flags STOF`
	Restores the original flag status.

`» 0 MENU`	Restores the previous menu. (You must first press CANCEL to clear the plot.)
`»`	
ENTER ' PIE STO	Stores the program in *PIE*.

Checksum: # 16631d
Bytes: 737

Example: The inventory at Fruit of the Vroom, a drive-in fruit stand, includes 983 oranges, 416 apples, and 85 bananas. Draw a pie chart to show each fruit's percentage of total inventory.

VAR PIE

Clear the current statistics data. (The prompt is removed from the display.) Key in the new data and draw the pie chart.

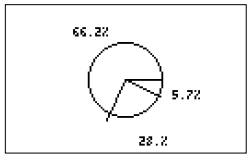
`CLEAR`

983 SLICE

416 SLICE

85 SLICE

DRAW



Press CANCEL to return to the stack display.

Checksum: # 1522d
Bytes: 189.5

Example: Convert the following algebraic expression to a series of objects in RPN syntax:

`'A*COS(B+√(C/D))-X^3'`.

' A × COS B + √x ← ( ) ___

C ÷ D ▶ ▶ − X y^x 3 ENTER →TAGR

Bessel Functions

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

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

Level 1	→	Level 1
z	→	*Ber(z)*

Techniques used in BER

- **Local variable structure.** At its outer level, *BER* consists solely of a local variable structure and so has two properties of a user-defined function: it can take numeric or symbolic arguments from the stack, or it can take arguments in algebraic syntax. However, because BER uses a DO…UNTIL…END loop, its defining procedure is a *program*. (Loop structures are not allowed in algebraic expressions.) Therefore, unlike user-defined functions, *BER* is not differentiable.
- **DO…UNTIL…END loop (indefinite loop with counter).** *BER* calculates successive terms in the series using a counter variable. When the new term does not differ from the previous term to within the 12-digit precision of the calculator, the loop ends.
- **Nested local variable structures.** The outer structure is consistent with the requirements of a user-defined function. The inner structure allows storing and recalling of key parameters.

Programmatic Use of Statistics and Plotting

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

Techniques used in PIE

- **Programmatic use of PLOT commands.** *PIE* executes XRNG and YRNG to define *x*- and *y*-axis display ranges in user units, and executes ARC and LINE to draw the circle and individual slices.
- **Programmatic use of matrices and statistics commands.**
- **Manipulating graphics objects.** *PIE* recalls *PICT* to the stack and executes GOR to merge the label for each slice with the plot.
- **FOR…NEXT (definite loop).** Each slice is calculated, drawn, and labeled in a definite loop.
- **CASE…END structure.** To avoid overwriting the circle, each label is offset from the midpoint of the arc of the slice. The offset for each label depends on the position of the slice in the circle. The CASE…END structure assigns an offset to the label based on the position of the slice.
- **Preserving calculator flag status.** Before specifying Radians mode, *PIE* saves the current flag status in a local variable, then restores that status at the end of the program.
- **Nested local variable structures.** At different parts of the process, intermediate results are saved in local variables for convenient recall as needed.
- **Temporary menu for data input.**

PIE program listing

Program:	Comments:
`«`	
`RCLF → flags`	Recalls the current flag status and stores it in variable *flags*.
`« RAD`	Sets Radians mode.
`{{ "SLICE" Σ+ } { } { "CLEAR" CLΣ } { } { } { "DRAW" CONT }}`	Defines the input menu: key 1 executes Σ+ to store each data point in Σ*DAT*, key 3 clears Σ*DAT*, and key 6 continues program execution after data entry.
`TMENU`	Displays the temporary menu.
`"Key values into SLICE,↵DRAW restarts program." PROMPT`	Prompts for inputs. ↵ represents the newline character ( ) after you enter the program on the stack.
`ERASE 1 131 XRNG 1 64 YRNG CLLCD`	Erases the current *PICT* and sets plot parameters.
`"Please wait...↵ Drawing Pie Chart" 1 DISP`	Displays "drawing" message.

Animation of Successive Taylor's Polynomials

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

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

SINTP (Converting a Plot to a Graphics Object)

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

Techniques used in SINTP

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

SINTP program listing

Program:	Comments:
`«`	
`'SIN(X)' STEQ`	Stores the expression for sin *x* in *EQ*.
`FUNCTION '-2*π' →NUM DUP NEG XRNG -2 2 YRNG`	Sets the plot type and *x*- and *y*-axis display ranges.
`ERASE DRAW`	Erases *PICT*, then plots the expression.
`PICT RCL 'SINT' STO`	Recalls the resultant graphics object and stores it in *SINT*.
`»`	
ENTER ' SINTP STO	Stores the program in *SINTP*.

Checksum: # 41184d
Bytes: 94

SINTP is demonstrated in the program *TSA*.

COL+

Type: Command

Description: Insert Column Command: Inserts an array (vector or matrix) into a matrix (or one or more elements into a vector) at the position indicated by n_{index}, and returns the modified array. The inserted array must have the same number of rows as the target array. n_{index} is rounded to the nearest integer. The original array is redimensioned to include the new columns or elements, and the elements at and to the right of the insertion point are shifted to the right.

Access: ← MTH MATRIX COL COL+ (MTH is the left-shift of the SYMB key).

← MATRICES CREATE COLUMN COL+ (MATRICES is the left-shift of the 5 key).

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[[matrix]]$_1$	*[[matrix]]*$_2$	n_{index}	→	*[[matrix]]*$_3$
[[matrix]]$_1$	*[vector]*$_{column}$	n_{index}	→	*[[matrix]]*$_2$
[vector]$_1$	$n_{element}$	n_{index}	→	*[vector]*$_2$

See also: COL–, CSWP, ROW+, ROW–

COLCT

Type: Command

Description: Factorizes a polynomial or an integer. Almost identical to COLLECT.

Access: ↱ CAT COLCT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*symb*$_1$'	→	'*symb*$_2$'
x	→	*x*
(x, y)	→	*(x, y)*

Example 1: `COLCT('5+X+9')` returns `'X+14'`

Example 2: `COLCT('X*1_m+X*9_cm')` returns `'X*1.09_m'`

Example 3: `COLCT('X^2*Y*X^T*Y')` returns `'Y^2*X^2*X^T'`

Example 4: `COLCT('X+3*X+Y+Y')` returns `'4*X+2*Y'`

See also: EXPAN, FACTOR, ISOL, QUAD, SHOW

COLLECT

Type: Command

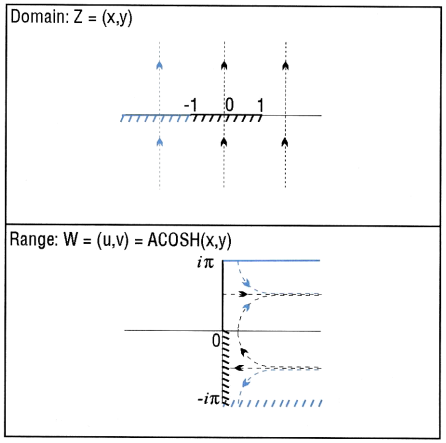
Description: Factorizes a polynomial or an integer. This command is almost identical to the COLCT command and similar to the FACTOR command. Unlike FACTOR it does not factorize symbolically into square roots. It is included to ensure backward-compatibility with earlier calculators.

Access: Algebra, ↱ ALG

Input: An expression or an integer

Output: The factorized expression, or the integer expressed as the product of prime numbers.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
If complex inputs are given, complex mode must be set (flag –103 set).



Branch Cut for ACOSH(Z)

ADD

Type: Command

Description: Add List Command: Adds corresponding elements of two lists or adds a number to each of the elements of a list.

ADD executes the + command once for each of the elements in the list. If two lists are the arguments, they must have the same number of elements as ADD will execute the + command once for each corresponding pair of elements. If one argument is a non-list object, ADD will attempt to execute the + command using the non-list object and each element of the list argument, returning the result to the corresponding position in the result. (See the + command entry to see the object combinations that are defined.) If an undefined addition is encountered, a Bad Argument Type error results.

Access: ← MTH LIST ADD (MTH is the left-shift of the SYMB key).

Flags: Binary Integer Wordsize (–5 through –10)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ *list*$_1$ }	{ *list*$_2$ }	→	{ *list*$_{result}$ }
{ *list* }	*obj*$_{non\text{-}list}$	→	{ *list*$_{result}$ }
obj$_{non\text{-}list}$	{ *list* }	→	{ *list*$_{result}$ }

See also: +, ΔLIST, ΠLIST, ΣLIST

ADDTMOD

Type: Function

Description: Adds two expressions or values, modulo the current modulus.

Access: Arithmetic, ← ARITH MODULO

Input: Level 2/Argument 1: The first expression.
Level 1/Argument 2: The second expression.

Output: The sum of the two expressions, modulo the current modulus.

Access: ⌐ CAT CLOSEIO
Input/Output: None
See also: BUFLEN, OPENIO

CLΣ

Type: Command
Description: Purges the current statistics matrix (reserved variable *ΣDAT*).
Access: ⌐ CAT CLΣ
Input/Output: None
See also: RCLΣ, STOΣ, Σ+, Σ–

CLUSR

Type: Command
Description: Clear Variables Command: Provided for compatibility with the HP 28 series. CLUSR is the same as CLVAR. See CLVAR.
Access: None. Must be typed in.

CLVAR

Type: Command
Description: Clear Variables Command: Purges all variables and empty subdirectories in the current directory.
Access: ⌐ CAT CLVAR
Input/Output: None
See also: PGDIR, PURGE

CMPLX

Type: Command
Description: Displays a menu of commands pertaining to complex numbers.
Access: ⌐ CAT CMPLX
Input/Output: None
See also: ARIT, DIFF, EXP&LN, SOLVER, TRIGO

CNRM

Type: Command
Description: Column Norm Command: Returns the column norm (one-norm) of the array argument.

The column norm of a matrix is the maximum (over all columns) of the sum of the absolute values of all elements in each column. For a vector, the column norm is the sum of the absolute values of the vector elements. For complex arrays, the absolute value of a given element (x, y) is $\sqrt{x^2 + y^2}$.

Access: ¬ MATRICES OPERATIONS CNRM (MATRICES is the left-shift of the 5 key).
Input/Output:

Level 1/Argument 1		Level 1/Item 1
[array]	→	$x_{\text{columnnoorm}}$

See also: CROSS, DET, DOT, RNRM

AMORT

Type: Command
Description: Amortize Command: Amortizes a loan or investment based upon the current amortization settings.

Values must be stored in the TVM variables (*I%YR*, *PV*, *PMT*, and *PYR*). The number of payments *n* is taken from the input together with flag –14.

Access: ⌐ & $.SLV TVM AMORT ($.SLV is the left-shift of the 7 key).
Flags: Financial Payment Mode (–14)
Input/Output:

Level 1/Argument 1		Level 3/Item 1	Level 2/Item 2	Level 1/Item 3
n	→	*principal*	*interest*	*balance*

See also: TVM, TVMBEG, TVMEND, TVMROOT

AND

Type: Function
Description: And Function: Returns the logical AND of two arguments.

When the arguments are binary integers or strings, AND does a bit-by-bit (base 2) logical comparison.

- An argument that is a binary integer is treated as a sequence of bits as long as the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments as shown in the following table.

bit_1	bit_2	bit_1 **AND** bit_2
0	0	0
0	1	0
1	0	0
1	1	1

- An argument that is a string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must have the same number of characters.

  When the arguments are real numbers or symbolics, AND simply does a true/false test. The result is 1 (true) if both arguments are non-zero; it is 0 (false) if either or both arguments are zero. This test is usually done to compare two test results.

  If either or both of the arguments are algebraic expressions, then the result is an algebraic of the form $symb_1$ AND $symb_2$. Execute →NUM (or set flag –3 before executing AND) to produce a numeric result from the algebraic result.

Access: ⌐ BASE NXT LOGIC AND (BASE is the right-shift of the 3 key).
Flags: Numerical Results (–3), Binary Integer Wordsize (–5 through –10)

The character codes are an extension of ISO 8859/1. Codes 128 through 160 are unique to the calculator. See Appendix J for a complete list of characters and character codes.

The default character ▪ is supplied for all character codes that are *not* part of the normal calculator's display character set.

Character code 0 is used for the special purpose of marking the end of the command line. Attempting to edit a string containing this character causes the error Can't Edit Null Char.

You can use the CHARS application to find the character code for any character used by the calculator. See "Additional Character Set" in Appendix D of the *HP 50g User's Guide*.

Access: ◀ PRG TYPE ▶ CHR (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	*"string"*

See also: NUM, POS, REPL, SIZE, SUB

CIRC

Type: Command

Description: Composes two permutations.

Access: Arithmetic, ◀ ARITH PERMUTATION

Input: Two lists, *L*1 and *L*2, representing two permutations. The composition *L*1○*L*2 is the permutation equivalent to performing permutation *L*2 first and *L*1 second.

Level 2/Argument 1: *L*1
Level 1/Argument 2: *L*2

Output: A list representing the single equivalent permutation, *L* = *L*1○*L*2

Example: Compose the permutations given by {3,4,5,2,1} and {2,1,4,3,5}

Command: CIRC({3,4,5,2,1},{2,1,4,3,5})

Result: {4,3,2,5,1}

See also: C2P, P2C

CKSM

Type: Command

Description: Checksum Command: Specifies the error-detection scheme.

Legal values for $n_{checksum}$ are as follows:

- 1-digit arithmetic checksum.
- 2-digit arithmetic checksum.
- 3-digit cyclic redundancy check (default).

The CKSM specified is the error-detection scheme that will be requested by KGET, PKT, or SEND. If the sender and receiver disagree about the request, however, then a 1-digit arithmetic checksum will be used.

Access: ▶ CAT CKSM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{checksum}$	→	

See also: BAUD, PARITY, TRANSIO

characters), as might have been left on the stack by entries when running in algebraic mode, will be ignored.

Access: ◀ ANS (ANS is the left-shift of the ENTER key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	obj_n

See also: LAST, LASTARG, PICK

APPLY

Type: Function

Description: Apply to Arguments Function: Creates an expression from the specified function name and arguments.

A user-defined function *f* that checks its arguments for special cases often can't determine whether a symbolic argument *x* represents one of the special cases. The function *f* can use APPLY to create a new expression *f(x)*. If the user now evaluates *f(x)*, *x* is evaluated before *f*, so the argument to *f* will be the result obtained by evaluating *x*.

When evaluated in an algebraic expression, APPLY evaluates the arguments (to resolve local names in user-defined functions) before creating the new object.

Access: ▶ CAT APPLY

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ $symb_1$ ... $symb_n$ }	'*name*'	→	'*name*($symb_1$ ... $symb_n$)'

Example: The following user-defined function *Asin* is a variant of the built-in function ASIN. *Asin* checks for special numerical arguments. If the argument on the stack is symbolic (the second case in the case structure), *Asin* uses APPLY to return the expression 'Asin(*argument*)'.

```
« → argument « CASE -3 FS? THEN argument ASIN END
  { 6 7 9 } argument TYPE POS THEN
  'APPLY(Asin,argument)' EVAL END
  'argument==1' THEN 'π/2' END
  'argument==-1' THEN '-π/2' END
  argument ASIN
  END » »
```

ENTER ' Asin STO▶

See also: QUOTE, |

ARC

Type: Command

Description: Draw Arc Command: Draws an arc in *PICT* counterclockwise from $x_{θ1}$ to $x_{θ2}$, with its center at the coordinate specified in argument 1 or level 4 and its radius specified in argument 2 or level 3.

ARC always draws an arc of constant radius in pixels, even when the radius and center are specified in user-units, regardless of the relative scales in user-units of the *x*- and *y*-axes. With user-unit arguments, the arc starts at the pixel specified by (*x*, *y*) + (*a*, *b*), where (*a*, *b*) is the rectangular conversion of the polar coordinate (x_{radius}, $x_{θ1}$). The resultant distance in pixels from the starting point to the center pixel is used as the actual radius, *r*'. The arc stops at the pixel specified by (*r*', $x_{θ2}$).

If $x_{θ1} = x_{θ2}$, ARC plots one point. If $|x_{θ1} - x_{θ2}| > 360$ degrees, 2π radians, or 400 grads, ARC draws a complete circle.

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	*y*	→	100(*y* – *x*)/*x*
x	'*symb*'	→	'%CH(*x*,*symb*)'
'*symb*'	*x*	→	'%CH(*symb*,*x*)'
'*symb*$_1$'	'*symb*$_2$'	→	'%CH(*symb*$_1$, *symb*$_2$)'
x_unit	*y_unit*	→	100(*y_unit* – *x_unit*)/*x_unit*
x_unit	'*symb*'	→	'%CH(*x_unit*,*symb*)'
'*symb*'	*x_unit*	→	'%CH(*symb*,*x_unit*)'

Example 1: `1_m 500_cm %CH` returns `400`, because 500 cm represents an increase of 400% over 1 m.

Example 2: `100_K 150_K %CH` returns `50`.

See also: %, %T

CHINREM

Type: Command

Description: Chinese Remainder function. Solves a system of simultaneous polynomial congruences in the ring Z[*x*].

Access: Arithmetic, ◀ ARITH POLYNOMIAL

Input: Level 2/Argument 1: A vector of the first congruence (expression and modulus).
Level 1/Argument 2: A vector of the second congruence (expression and modulus).

Output: A vector of the solution congruence (expression and modulus).

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Solve the following simultaneous congruences for the polynomial *u*:

$u \equiv x^2 + 1 \pmod{x+2}$

$u \equiv x - 1 \pmod{x+3}$

Command: `CHINREM([X^2+1,X+2],[X-1,X+3])`

Result: `[X^3+2*X^2+5,-(X^2+5*X+6)]`

See also: EGCD, ICHINREM

CHOLESKY

Type: Command

Description: Returns the Cholesky factorization of a square matrix.

Access: Matrices, ◀ MATRICES QUADRATIC FORM

Input: A positive square matrix, *M*

Output: An upper triangular matrix, *P*, such that $^tP*P=M$. (^tP is the transpose of P.)

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

ARIT

Type: Command

Description: Displays a menu or list showing the three CAS submenus for arithmetical operations, INTEGER, MODULAR and POLYNOMIAL.

Access: Catalog, ▶ CAT

Flags: If the CHOOSE boxes flag is clear (flag –117 clear), displays the submenus as a numbered list. If the flag is set, displays the operations as a menu of function keys.

See also: ALGB, CONSTANTS, DIFF, EXP&LN, INTEGER, MAIN, MATHS, MATR, MODULAR, POLYNOMIAL, REWRITE, TESTS, TRIGO

ARRY→

Type: Command

Description: Array to Stack Command: Takes an array and returns its elements as separate real or complex numbers. Also returns a list of the dimensions of the array.

If the argument is an *n*-element vector, the first element is returned to level *n* + 1 (not level *nm* + 1), and the *n*th element to level 2.

Access: ▶ CAT ARRY→

Input/Output:

Level 1/Argument 1		Lnm+1/A1 ... L2/Anm	Level1/Itemnm+1
[vector]	→	$z_1 \ldots z_n$	{ $n_{element}$ }
[[matrix]]	→	$z_{11} \ldots z_{nm}$	{ n_{row} m_{col} }

L = Level; I = item

See also: →ARRY, DTAG, EQ→, LIST→, OBJ→, STR→

→ARRY

Type: Command

Description: Stack to Array Command: Returns a vector of *n* real or complex elements or a matrix of *n* × *m* real or complex elements.

The elements of the result array should be entered in row order. If one or more of the elements is a complex number, the result array will be complex.

Access: ◀ PRG TYPE →ARRY (PRG is the left-shift of the EVAL key).

Input/Output:

Levelnm+1/Argument1 ... Level2/Argumentnm	Level1/Argumentnm+1		Level1/Item1
$z_1 \ldots z_n$	$n_{element}$	→	*[vector]*
$z_{11} \ldots z_{n,m}$	{ n_{row}, m_{col} }	→	*[[matrix]]*

See also: ARRY→, LIST→, →LIST, OBJ→, STR→, →TAG, →UNIT

ASIN

Type: Analytic Function

Description: Arc Sine Analytic Function: Returns the value of the angle having the given sine.

For a real argument *x* in the domain $-1 \le x \le 1$, the result ranges from –90 to +90 degrees ($-\pi/2$ to $+\pi/2$ radians; –100 to +100 grads).

A real argument outside of this domain is converted to a complex argument $z = x + 0i$, and the result is complex.

CASCMD

Type: Command

Description: Displays a list of CAS operations. Selecting one with OK displays a description, related operations, an example of the operation, and the option to copy the example to the command line. More details are given in Appendix C and Appendix H of the User's Guide. If level 1 of the stack contains a string, the list of CAS operations will be displayed beginning at this point.

Access: Catalog, ↱ CAT, or tools TOOL NXT

See also: HELP

CASE

Type: Command

Description: CASE Conditional Structure Command: Starts CASE … END conditional structure.

The CASE … END structure executes a series of *cases* (tests). The first test that returns a true result causes execution of the corresponding true-clause, ending the CASE … END structure. A default clause can also be included: this clause executes if all tests evaluate to false. The CASE command is available in RPN programming only. You cannot use it in algebraic programming.

The CASE … END structure has this syntax:

CASE
test-clause$_1$ THEN *true-clause*$_1$ END
test-clause$_2$ THEN *true-clause*$_2$ END
.
.
test-clause$_n$ THEN *true-clause*$_n$ END
default-clause (*optional*)
END

When CASE executes, *test-clause*$_1$ is evaluated. If the test is true, *true-clause*$_1$ executes, then execution skips to END. If *test-clause*$_1$ is false, *test-clause*$_2$ executes. Execution within the CASE structure continues until a true clause is executed, or until all the test clauses evaluate to false. If the default clause is included, it executes if all test clauses evaluate to false.

Access: ↰ PRG BRCH CASE (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
CASE	→	
THEN	*T/F* →	
END	→	
END	→	

Example: The following program takes a numeric argument from the stack:

- if the argument is negative, it is added to itself
- if the argument is positive, it is negated
- if the argument is zero, the program aborts

```
« → X « CASE
 'X>0' THEN X NEG END
 'X<0' THEN X DUP + END
 'X==0' THEN 0 DOERR END
 END » »
```

See also: END, IF, IFERR, THEN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*asin z*
'*symb*'	→	'*ASIN(symb)*'

See also: ACOS, ATAN, ISOL, SIN

ASIN2C

Type: Command

Description: Transforms an expression by replacing asin(*x*) subexpressions with π/2–acos(*x*) subexpressions.

Access: Trigonometry, ↱ TRIG

Input: An expression

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

See also: ACOS2S, ASIN2T, ATAN2S

ASIN2T

Type: Command

Description: Transforms an expression by replacing asin(*x*) subexpressions with the following:

$$\text{atan}\left(\frac{x}{\sqrt{1-x^2}}\right)$$

Access: Trigonometry, ↱ TRIG

Input: An expression.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

See also: ASIN2C, ACOS2S, ATAN2S

ASINH

Type: Analytic Function

Description: Arc Hyperbolic Sine Analytic Function: Returns the inverse hyperbolic sine of the argument.

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

```
ASINH(Z)*(-1)^n1+π*i*n1
```

The function ASINH is the inverse of a *part* of SINH, a part defined by restricting the domain of SINH such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

The points in this restricted domain of SINH are called the *principal values* of the inverse relation. ASINH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ASINH to the boundary of the restricted domain of SINH form the *branch cuts* of ASINH.

Input/Output:

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

See also: ARC, LINE, TLINE

BUFLEN

Type: Command

Description: Buffer Length Command: Returns the number of characters in the calculator's serial input buffer and a single digit indicating whether an error occurred during data reception.

The digit returned is 1 if no framing, UART overrun, or input-buffer overflow errors occurred during reception, or 0 if one of these errors did occur. (The input buffer holds up to 255 bytes.) When a framing or overrun error occurs, data reception ceases until the error is cleared (which BUFLEN does); therefore, *n* represents the data received *before* the error.

Use ERRM to see which error has occurred when BUFLEN returns 0 to level 1.

Access: ⌐ CAT BUFLEN

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
	→	n_{chars}	0/1

See also: CLOSEIO, OPENIO, SBRK, SRECV, STIME, XMIT

BYTES

Type: Command

Description: Byte Size Command: Returns the number of bytes and the checksum for the given object.

If the argument is a built-in object, then the size is 2.5 bytes and the checksum is #0.

If the argument is a global name, then the size represents the name *and* its contents, while the checksum represents the contents only. The size of the name alone is (3.5 + *n*), where *n* is the number of characters in the name.

Access: ← PRG MEMORY BYTES (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
obj	→	#$n_{checksum}$	x_{size}

Example: Objects that decompile identically can have different byte sizes and checksums. For instance,

```
{1}
```

and

```
1 'A' STO A {} +
```

both produce lists containing the number 1. However, the first list contains the built-in object 1 (for a size of 7.5 bytes), while the second list contains a RAM copy of 1 (for a size of 15.5 bytes).

See also: MEM

B→R

Type: Command

Description: Binary to Real Command: Converts a binary integer to its floating-point equivalent.

Be careful not to reassign or suppress the keys necessary to cancel User mode. If this happens, exit User mode by doing a system halt ("warm start"): press and hold ON and F3 simultaneously, releasing F3 first. This cancels User mode.

Access: ⌐ CAT ASN OR ← &MODE KEYS ASN

Flags: User-Mode Lock (–61) and User Mode (–62) affect the status of the user keyboard

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	x_{key}	→	
'*SKEY*'	x_{key}	→	

Example: Executing ASN with GETI in level 2 and 75.3 in level 1 assigns GETI to ⌐ "" on the user keyboard. (⌐ "" has a location of 75.3 because it is seven rows down, five columns across, and right-shifted.) When the calculator is in User mode, pressing ⌐ "" now executes GETI (instead of executing "").

See also: DELKEYS, RCLKEYS, STOKEYS

ASR

Type: Command

Description: Arithmetic Shift Right Command: Shifts a binary integer one bit to the right, except for the most significant bit, which is maintained.

The most significant bit is preserved while the remaining (*wordsize* –1) bits are shifted right one bit. The second-most significant bit is replaced with a zero. The least significant bit is shifted out and lost.

An arithmetic shift is useful for preserving the sign bit of a binary integer that will be shifted. Although the calculator makes no special provision for signed binary integers, you can still *interpret* a number as a signed quantity.

Access: ⌐ BASE NXT BIT ASR (BASE is the right-shift of the 3 key).

Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n_1	→	#n_2

See also: SL, SLB, SR, SRB

ASSUME

Type: Function

Description: Adds global names to the reserved variable REALASSUME, with specific assumptions. REALASSUME is a list of the global variables that will be considered by some CAS operations to represent *real* numbers when complex mode is set. ASSUME adds further assumptions, for example that a variable is greater than or equal to zero. Assumptions must be of the form *v*≤expression, or *v*≥expression, where *v* is the variable name. Several assumptions can be combined.

These assumptions are used by the solve commands; for example if a variable is assumed to be greater than zero then the solvers will not look for solutions where that variable is negative. Some of the solvers will give complex solutions for variables even if they are in REALASSUME.

Access: Catalog, ⌐ CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{baudrate}$	→	

See also: CKSM, PARITY, TRANSIO

BEEP

Type: Command

Description: Beep Command: Sounds a tone at *n* hertz for *x* seconds.

The frequency of the tone is subject to the resolution of the built-in tone generator. The minimum frequency is 1 Hz and the maximum frequency in 15000 Hz. An input that doesn't round to an integer within this range will cause the BEEP command to be skipped. Durations greater than 1200 seconds are automatically changed to 1200 seconds.

Access: ◀┐ PRG NXT OUT NXT BEEP (PRG is the left-shift of the EVAL key).

Flags: Error Beep (–56)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$n_{frequency}$	$x_{duration}$	→	

See also: HALT, INPUT, PROMPT, WAIT

BESTFIT

Type: Command

Description: Best-Fitting Model Command: Executes LR with each of the four curve fitting models, and selects the model yielding the largest correlation coefficient.

The selected model is stored as the fifth parameter in the reserved variable *ΣPAR*, and the associated regression coefficients, intercept and slope, are stored as the third and fourth parameters, respectively.

Access: ┌▶ CAT BESTFIT

Input/Output: None

See also: EXPFIT, LINFIT, LOGFIT, LR, PWRFIT

BIN

Type: Command

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

Binary integers require the prefix #. Binary integers entered and returned in binary base automatically show the suffix b. If the current base is not binary, binary numbers can still be entered by using the suffix b (the numbers are displayed in the current base, however).

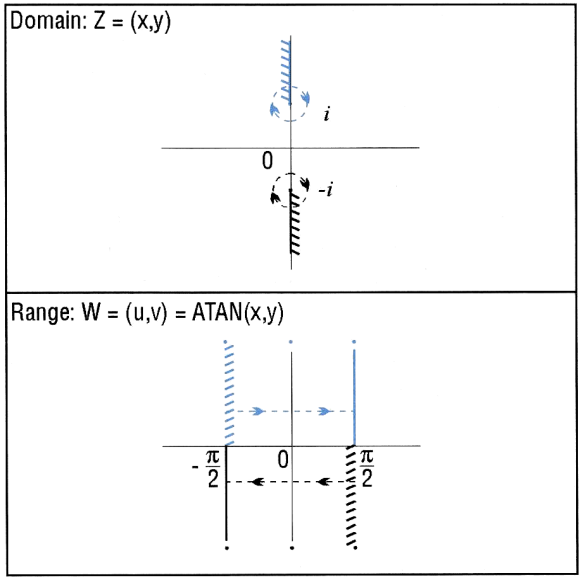
The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Access: ┌▶ BASE BIN (BASE is the right-shift of the 3 key).

Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12)

Input/Output: None

See also: DEC, HEX, OCT, STWS, RCWS



Branch Cuts for ATAN(Z)

Access: ◀┐ ATAN (ATAN is the left-shift of the TAN key).

Flags: Principal Solution (–1), Numerical Results (–3), Angle Mode (–17, –18)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*atan z*
'symb'	→	*'ATAN(symb)'*

See also: ACOS, ASIN, ISOL, TAN

ATAN2S

Type: Command

Description: Transforms an expression by replacing atan(*x*) subexpressions with the following:

$$\text{asin}\left(\frac{x}{\sqrt{x^2+1}}\right)$$

Access: Trigonometry, ┌▶ TRIG

Input: An expression.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

See also: ASIN2C, ACOS2S, ASIN2T

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

See also: AXL, AXQ

AXQ

Type: Command

Description: Converts a square matrix into the associated quadratic form.

Access: Convert, ← CONVERT MATRIX CONVERT, or matrices ← MATRICES QUADRATIC FORM

Input: Level 2/Argument 1: An $n \times n$ matrix.
Level 1/Argument 2: A vector containing n variables.

Output: Level 2/Item 1: The corresponding quadratic form.
Level 1/Item 2: The vector containing the variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Find the quadratic form, expressed in terms of x, y, and z, associated with the following matrix:

$$\begin{bmatrix} 3 & 6 & 0 \\ 2 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Command:
```
AXQ([[3,6,0][2,4,1][1,1,1]],[X,Y,Z])
```
Result:
```
{3*X^2+(8*Y+Z)*X+(4*Y^2+2*Z*Y+Z^2),[X,Y,Z]}
```

See also: AXL, AXM, GAUSS, QXA

BAR

Type: Command

Description: Bar Plot Type Command: Sets the plot type to BAR.

When the plot type is BAR, the DRAW command plots a bar chart using data from one column of the current statistics matrix (reserved variable *ΣDAT*). The column to be plotted is specified by the XCOL command, and is stored in the first parameter of the reserved variable Σ*PAR*. 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 BAR, 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) for the HP 48gII and (–6.5,–3.9) for the HP 50g and 49g+.
- (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) for the HP 48gII and (6.5,4.0) for the HP 50g and 49g+.
- *indep* is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers, with the smaller of the numbers specifying the horizontal location of the first bar. The default value of *indep* is *X*.
- *res* is a real number specifying the bar width in user-unit coordinates, or a binary integer specifying the bar width in pixels. The default value is 0, which specifies a bar width of 1 in user-unit coordinates.
- *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).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	
#n	→	
{ x y }	→	
{ #n #m }	→	

See also: AXES, DRAX

ATTACH

Type: Command

Description: Attach Library Command: Attaches the library with the specified number to the current directory. Each library has a unique number. If a port number is specified, it is ignored.

To use a library object, it must be in a port and it must be attached. A library object copied into RAM (such as through the PC Link) must be stored into a port using STO.

Some libraries require you to ATTACH them.

You can ascertain whether a library is attached to the current directory by executing LIBS.

The number of libraries that can be attached to the HOME directory is limited only by the available memory. However, only one library at a time can be attached to any other directory. If you attempt to attach a second library to a non-*HOME* directory, the new library will overwrite the old one.

Access: → CAT ATTACH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{library}$	→	
$:n_{port}:n_{library}$	→	

See also: DETACH, LIBS

AUGMENT

Type: Command

Description: Concatenate two lists, a list and an element, or a vector and an element. Also creates a matrix from component row vectors.

Access: Matrices, ← MATRICES CREATE

Input: Level 2/Argument 1: A vector, a list, a matrix, or a string.
Level 1/Argument 2: A vector, a list, a matrix, or an element.

Output: The matrix, list or string formed by combining the arguments. In the case of a string in level 2, AUGMENT acts exactly like "+" or "ADD".

Example 1: Append 3 to the list {1,2}:

Command:
```
AUGMENT({1,2},3)
```
Result:
```
{1,2,3}
```

Example 2: Combine the rows [1,2,3] and [4,5,6] into a matrix:

Command:
```
AUGMENT([1,2,3],[4,5,6])
```

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

The number returned for *e* is the closest approximation 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. Even though the calculator often *displays* 'EXP(*x*)' as *e*^*x*, it's still 'EXP(*x*)' internally.

Access: ALPHA ◀ E

◀ MTH NXT CONSTANTS *e* (MTH is the left-shift of the SYMB key).

◀ MTH NXT CONSTANTS 2.718281828… (MTH is the left-shift of the SYMB key).

Flags: Symbolic Constants (–2), Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	'*e*'
	→	*2.71828182846*

See also: EXP, EXPM, *i*, LN, LNP1, MAXR, MINR, π

EDIT

Type: Command

Description: Edit Command: Moves specified object to the command line where it can be edited.

Access: ◀ ▼

TOOL ◀ EDIT

Input/Output: None

See also: EDITB, VISIT

EDITB

Type: Command

Description: Edit Best Command: Opens the specified object in the most suitable editor. For example, if you use a matrix as the specified object, the command opens it in Matrix Writer.

Access: ▼

TOOL EDIT

Input/Output: None

See also: EDIT, VISIT

EGCD

Type: Command

Description: Given two polynomials, *a* and *b*, returns polynomials *u*, *v*, and *c* where: $au+bv=c$
In the equation, *c* is the greatest common divisor of *a* and *b*.

Access: Arithmetic, ◀ ARITH POLYNOMIAL

Input: Level 2/Argument 1: The expression corresponding to *a* in the equation.
Level 1/Argument 2: The expression corresponding to *b* in the equation.

Output: Level 3/Item 1: The result corresponding to *c* in the equation.
Level 2/Item 2: The result corresponding to *u* in the equation.
Level 1/Item 3: The result corresponding to *v* in the equation.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Factorize the following:
$x^2 + 5x + 6$

Command: `COLLECT(X^2+5*X+6)`

Result: `(X+2)(X+3)`

See also: COLCT, EXPAND, FACTOR

COLΣ

Type: Command

Description: Column Sigma Command: Specifies the independent-variable and dependent-variable columns of the current statistics matrix (the reserved variable *ΣDAT*).
COLΣ combines the functionality of XCOL and YCOL. The independent-variable column number x_{xcol} is stored as the first parameter in the reserved variable *ΣPAR* (the default is 1). The dependent-variable column number x_{ycol} is stored as the second parameter in the reserved variable *ΣPAR* (the default is 2).
COLΣ accepts and stores noninteger values, but subsequent commands that use these two parameters in *ΣPAR* will cause errors.

Access: ▶ CAT COLΣ

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{col}	y_{col}	→	

Example: `2 5 COLΣ` sets column 2 in *ΣDAT* as the independent-variable column, sets column 5 as the dependent-variable column, and stores 2 and 5 as the first and second elements in *ΣPAR*.

See also: BARPLOT, BESTFIT, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL, YCOL

COMB

Type: Function

Description: Combinations Function: Returns the number of possible combinations of *n* items taken *m* at a time. The following formula is used:

$$C_{n,m} = \frac{n!}{m! \cdot (n-m)!}$$

The arguments *n* and *m* must each be less than 10^{12}. If $n<m$, zero is returned.

Access: ◀ MTH NXT PROBABILITY COMB (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	*m*	→	$C_{n,m}$
'*symb*$_n$'	*m*	→	'*COMB(symb*$_n$*,m)*'
n	'*symb*$_m$'	→	'*COMB(n, symb*$_m$*)*'
'*symb*$_n$'	'*symb*$_m$'	→	'*COMB(symb*$_n$*,symb*$_m$*)*'

See also: FACT, PERM, !

CON

Type: Command

Description: Constant Array Command: Returns a constant array, defined as an array whose elements all have the same value.

Input/Output:

Level 2	Level 1		Level 1
obj$_1$	*obj*$_2$	→	

See also: CLEAR, DROP, DROPN

DROPN

Type: RPL Command

Description: Drop n Objects Command: Removes the first *n* + 1 objects from the stack (the first *n* objects excluding the integer *n* itself).

Access: ◀ PRG STACK NXT NXT DROPN (PRG is the left-shift of the EVAL key).
TOOL STACK NXT NXT DROPN

Input/Output:

Level$_{n+1}$ ... Level 2	Level 1		Level 1
obj1 ... obj$_n$	*n*	→	

See also: CLEAR, DROP, DROP2

DTAG

Type: Command

Description: Delete Tag Command: DTAG removes all tags (labels) from an object.
The leading colon is not shown for readability when the tagged object is on the stack.
DTAG has no effect on an untagged object.

Access: ◀ PRG TYPE NXT DTAG (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
tag:obj	→	*obj*

See also: LIST→, →TAG

DUP

Type: RPL Command

Description: Duplicate Object Command: DUP returns a copy of the argument (or the object on level 1).

Access: ◀ PRG STACK DUP (PRG is the left-shift of the EVAL key).
TOOL STACK DUP
ENTER in RPN mode executes DUP when no command line is present.

Input/Output:

Level 1		Level 2	Level 1
obj	→	*obj*	*obj*

See also: DUPN, DUP2, PICK

DUP2

Type: RPL Command

Description: Duplicate 2 Objects Command: DUP2 returns copies of the two objects on levels 1 and 2 of the stack.

Access: ◀ PRG STACK NXT NXT DUP2 (PRG is the left-shift of the EVAL key).
TOOL STACK NXT NXT DUP2

Example: The following program computes the above rule of thumb for the number of accurate digits:

```
« DUP SIZE 1 GET LOG SWAP COND LOG + 11 SWAP - »
```

See also: SNRM, SRAD, TRACE

CONIC

Type: Command

Description: Conic Plot Type Command: Sets the plot type to CONIC.

When the plot type is CONIC, the DRAW command plots the current equation as a second-order polynomial of two real variables. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

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

For plot type CONIC, 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) for the HP 48gII and (–6.5,–3.9) for the HP 50g and 49g+.
- (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) for the HP 48gII and (6.5,4.0) for the HP 50g and 49g+.
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value is *X*.
- *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 complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes, or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command CONIC places the command name CONIC in *PPAR*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

The current equation is used to define a pair of functions of the independent variable. These functions are derived from the second-order Taylor's approximation to the current equation. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the values in (x_{min}, y_{min}) and (x_{max}, y_{max}) (the display range) are used. Lines are drawn between plotted points unless flag –31 is set.

Access: ▶ CAT CONIC

Input/Output: None

See also: BAR, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

CONJ

Type: Function

Description: Conjugate Analytic Function: Conjugates a complex number or a complex array.

Conjugation is the negation (sign reversal) of the imaginary part of a complex number. For real numbers and real arrays, the conjugate is identical to the original argument.

Access: ▶ CMPLX CONJ (CMPLX is the right-shift of the 1 key).

Flags: Numerical Results (–3)

DRAW

Type: Command Operation

Description: Draw Plot Command: Plots the mathematical data in the reserved variable *EQ* or the statistical data in the reserved variable *ΣDAT*, using the specified *x*- and *y*-axis display ranges.

The plot type determines if the data in the reserved variable *EQ* or the data in the reserved variable *ΣDAT* is plotted.

DRAW does not erase *PICT* before plotting; execute ERASE to do so. DRAW does not draw axes; execute DRAX to do so.

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

Access: [→] [CAT] DRAW

Flags: Simultaneous or Sequential Plot (–28), Curve Filling (–31)

Input/Output: None

See also: AUTO, AXES, DRAX, ERASE, FREEZE, PICTURE, LABEL, PVIEW

DRAW3DMATRIX

Type: Command

Description: Draws a 3D plot from the values in a specified matrix.

The number of rows indicates the number of units along the *x* axis, the number of columns indicates the number of units along the *y* axis, and the values in the matrix give the magnitudes of the plotted points along the *z* axis. In other words, the coordinates of a plotted point are (*r*, *c*, *v*) where *r* is the row number, *c* the column number and *v* the value in the corresponding cell of the matrix.

You can limit the points that are plotted by specifying a minimum value (v_{min}) and a maximum value (v_{max}). Values in the matrix outside this range are not plotted. If all values are included, the total number of points plotted is $r \times c$.

Once the plot has been drawn, you can rotate it in various ways by pressing the following keys:

(▶) and (◀) rotate the plot around the *x* axis (in different directions)
(▲) and (▼) rotate the plot around the *y* axis (in different directions)
[TOOL] and [NXT] rotate the plot around the *z* axis (in different directions)

Access: [→] [CAT] DRAW3DMATRIX

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[[*matrix*]]	v_{min}	v_{max}	→	

See also: FAST3D

DRAX

Type: Command

Description: Draw Axes Command: Draws axes in *PICT*.

The coordinates of the axes intersection are specified by AXES. Axes tick-marks are specified in *PPAR* with the ATICK, or AXES command. DRAX does not draw axes labels; execute LABEL to do so.

Access: [→] [CAT] DRAX

Input/Output: None

See also: AXES, DRAW, LABEL

CONT

Type: Command

Description: Continue Program Execution Command: Resumes execution of a halted program.

Since CONT is a command, it can be assigned to a key or to a custom menu.

Access: [→] [CONT] ([CONT] is the left-shift of the [ON] key).

Input/Output: None

Example: The program

```
« "Enter A, press ‹ CONT ›" ‹ CONT › MENU PROMPT »
```

displays a prompt message, builds a menu with the CONT command assigned to the first menu key, and halts the program for data input. After entering data, pressing [CONT] resumes program execution. (Note that pressing [→] [CONT] is equivalent to pressing [CONT].)

See also: HALT, KILL, PROMPT

CONVERT

Type: Command

Description: Convert Units Command: Converts a source unit object to the dimensions of a target unit.

The source and target units must be compatible. The number part x_2 of the target unit object is ignored.

Access: [→] [CONVERT] UNITS TOOLS CONVERT ([CONVERT] is the left-shift of the [6] key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$x_1\_units_{source}$	$x_2\_units_{target}$	→	$x_3\_units_{target}$

See also: UBASE, UFACT, →UNIT, UVAL

CORR

Type: Command

Description: Correlation Command: Returns the correlation coefficient of the independent and dependent data columns in the current statistics matrix (reserved variable *ΣDAT*).

The columns are specified by the first two elements in the reserved variable *ΣPAR*, set by XCOL and YCOL, respectively. If *ΣPAR* does not exist, CORR creates it and sets the elements to their default values (1 and 2).

The correlation is computed with the following formula:

$$\frac{\sum_{i=1}^{n}(x_{in_1}-\bar{x}_{n_1})(x_{in_2}-\bar{x}_{n_2})}{\sqrt{\sum_{i=1}^{n}(x_{in_1}-\bar{x}_{n_1})^2\sum_{i=1}^{n}(x_{in_2}-\bar{x}_{n_2})^2}}$$

where x_{in_1} is the *i*th coordinate value in column n_1, x_{in_2} is the *i*th coordinate value in the column n_2, $\bar{x}_{n_1}$ is the mean of the data in column n_1, $\bar{x}_{n_2}$ is the mean of the data in column n_2, and *n* is the number of data points.

Access: [→] [CAT] CORR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{correlation}$

See also: COLΣ, COV, PREDX, PREDY, XCOL, YCOL

Input/Output:

L_{n+3}/A_1 ... L_3/A_{n-2}	L_2/A_{n+1}	L_1/A_{n+2}		Level 1/Item 1
{ *list* }$_1$ ... { *list* }$_n$	*n*	« *program* »	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*command*	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*name*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	« *program* »	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	*command*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	*name*	→	{ *results* }

L = Level; A = Argument

Example: { 1 2 3 } { 4 5 6 } { 7 8 9 } 3 « + * » DOLIST returns { 11 26 45 }.

See also: DOSUBS, ENDSUB, NSUB, STREAM

DOMAIN

Type: Command

Description: For a function of the current variable, lists the domains of real numbers for which the function is defined and for which it is undefined. DOMAIN works for functions of more than one argument, for example DOMAIN (X*X), and for user defined functions, as in the example below. For functions which it does not recognize, DOMAIN returns the message "Unknown operator".

Access: Catalog, ↱ CAT

Input: Level 1/Item 1: A function, or an expression, in terms of the current variable.

Output: Level 1/Item 1: A list with regions where the function is undefined marked by '?' and regions where the function is defined marked by +. Rational singularities, such as 0 in 1/*x*, are not listed.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Define a function $f = \sqrt{(a+1)}$ by typing DEF(F(A)=√(A+1)). Then tabulate the domain over which it is defined and undefined.

Command: DOMAIN(F(X))

Result: {'-∞' '?' –1 + '+∞'}, showing that the function *f* is undefined for values from –∞ to –1 and is defined from –1 to +∞.

See also: SIGNTAB, TABVAR

DOSUBS

Type: Command

Description: Do to Sublist Command: Applies a program or command to groups of elements in a list.

The real number *n* can be omitted when the first argument is one of the following:

- A command.
- A user program containing a single command.
- A program with a user-defined function structure.
- A global or local name that refers to one of the above.

The first iteration uses elements 1 through *n* from the list; the second iteration uses elements 2 through *n* + 1; and so on. In general, the m^{th} iteration uses the elements from the list corresponding to positions *m* through *m* + *n* – 1.

During an iteration, the position of the first element used in that iteration is available to the user using the command NSUB, and the number of groups of elements is available using the

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$\Sigma_{covariance}$

See also: COLΣ, CORR, PCOV, PREDX, PREDY, XCOL, YCOL

CR

Type: Command

Description: Carriage Right Command: Prints the contents, if any, of the printer buffer.

When printing to the serial port (flag –34 set), CR sends to the printer a string that encodes the line termination method. The default termination method is carriage-return/linefeed. The string is the fourth parameter in the reserved variable *PRTPAR*.
When using the HP 82240B Infrared Printer (flag –34 clear), CR leaves the printhead on the right end of the just printed line.

Access: ↱ CAT CR

Flags: I/O Device (–33), Printing Device (–34), Double-Spaced Printing (–37), I/O Device for Wire (–78)

Input/Output: None

See also: DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

CRDIR

Type: Command

Description: Create Directory Command: Creates an empty subdirectory with the specified name in the current directory.

CRDIR does not change the current directory; evaluate the name of the new subdirectory to make it the current directory.

Access: ↰ PRG MEMORY DIRECTORY CRDIR(PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*global*'	→	

See also: HOME, PATH, PGDIR, UPDIR

CROSS

Type: Command

Description: Cross Product Command: CROSS returns the cross product C = A × B of vectors A and B.

The arguments must be vectors having two or three elements, and need not have the same number of elements. (The calculator automatically converts a two-element argument [d_1 d_2] to a three-element argument [d_1 d_2 0].)

Access: ↰ MTH VECTOR CROSS (MTH is the left-shift of the SYMB key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[*vector*]$_A$	[*vector*]$_B$	→	[*vector*]$_{A \times B}$

See also: CNRM, DET, DOT, RNRM

CSWP

Type: Command

Description: Column Swap Command: Swaps columns *i* and *j* of the argument matrix and returns the modified matrix, or swaps elements *i* and *j* of the argument vector and returns the modified vector.

Column numbers are rounded to the nearest integer. Vector arguments are treated as row vectors.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear). Radians mode must be set (flag –17 set).
Incremental power mode must be set (flag –114 set).

Example: Find the fourth degree Taylor polynomial for the following:

$$\frac{x^3 + 4x + 12}{11x^{11} + 1}$$

Command: `DIVPC(X^3+4*X+12,11*X^11+1,4)`

Result: `12+4*X+X^3`

See also: TAYLOR0, TAYLR, SERIES

dn

Type: Function

Description: Differential of a function with respect to its argument *n*. For example d1f(x,y) is the differential of f(x,y) with respect to x and d3g(y,z,t) is the differential of g(y,z,t) with respect to t. The second-order derivative of f(x,y) with respect to x is written d1d1f(x,y). The d*n* function is an alternative to the ∂ function; d1f(x,y) is the same as ∂x(f(x,y)). d*n* does not require brackets after it, it must be followed immediately by the function name, with no spaces. d*n* differentiates with respect to the whole of argument *n*, see the example. d*n* is mainly used for formal arguments, see the example in DESOLVE, but can be used to differentiate expressions, as in the example.

Access: Access is by typing the letter "d" from the alpha keyboard, followed by the number *n*, before the function whose differential is required.

Output: d*n* does not change its argument, it works like the negative sign placed before a number or an expression. If the argument can be differentiated, EVAL will carry out the differentiation.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Differentiate the function sin(2x) with respect to its argument:

Command: `EVAL(d1SIN(2*X))`

Result: `COS(2*X)`

(Note that the function was differentiated with respect to its argument 2x, not with respect to the variable x.)

See also: DERIV, DERVX, DESOLVE, ∂

DO

Type: Command

Description: DO Indefinite Loop Structure Command: Starts DO…UNTIL…END indefinite loop structure. DO … UNTIL … END executes a loop repeatedly until a test returns a true (nonzero) result. Since the test clause is executed after the loop clause, the loop is always executed at least once. The syntax is: DO *loop-clause* UNTIL *test-clause* END
DO starts execution of the loop clause. UNTIL ends the loop clause and begins the test clause. The test clause must return a test result to the stack. END removes the test result from the stack. If its value is zero, the loop clause is executed again; otherwise execution resumes following END.

Access: ← PRG BRANCH DO (PRG is the left-shift of the EVAL key).

Access: ← MTH VECTOR NXT CYLIN (MTH is the left-shift of the SYMB key).

Input/Output: None

See also: RECT, SPHERE

C→PX

Type: Command

Description: Complex to Pixel Command: Converts the specified user-unit coordinates to pixel coordinates. The user-unit coordinates are derived from the (x_{min}, y_{min}) and (x_{max}, y_{max}) parameters in the reserved variable *PPAR*.

Access: ← PRG NXT PICT NXT C→PX (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x, y)	→	{ #*n*, #*m* }

See also: PX→C

C→R

Type: Command

Description: Complex to Real Command: Separates the real and imaginary parts of a complex number or complex array. The result in item 1/level 2 represents the real part of the complex argument. The result in item 2/ level 1 represents the imaginary part of the complex argument.

Access: ← PRG TYPE NXT C→R (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
(x, y)	→	*x*	*y*
[C-array]	→	*[R-array]*$_1$	*[R-array]*$_2$

See also: R→C, RE, IM

DARCY

Type: Function

Description: Darcy Friction Factor Function: Calculates the Darcy friction factor of certain fluid flows. DARCY calculates the Fanning friction factor and multiplies it by 4. $x_{\varepsilon/D}$ 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 ($Re > 2100$). $x_{\varepsilon/D}$ and y_{Re} must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Access: → CAT DARCY

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_ε / *D*	y_{Re}	→	x_{Darcy}

See also: FANNING

DATE

Type: Command

Description: Date Command: Returns the system date.

Access: → TIME TOOLS DATE (TIME is the right-shift of the 9 key).
→ & 9 DATE

DIV

Type: Command

Description: Returns the divergence of a vector function.

Access: Calculus, ◁┐ CALC DERIV. & INTEG.

Input: Level 2/Argument 1: An array representing a vector function.
Level 1/Argument 2: An array containing the variables.

Output: The divergence of the vector function with respect to the specified variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Find the divergence of the following vector function:

$$v = x^2y\underset{\sim}{i} + x^2y\underset{\sim}{j} + y^2z\underset{\sim}{k}$$

Command: `DIV([X^2*Y, X^2*Y, Y^2*Z],[X,Y,Z])`

Result: `Y*(2*X)+(X^2+Y^2)`

See also: CURL, HESS

DIV2

Type: Command

Description: Performs Euclidean division on two expressions. Step-by-step mode is available with this command.

Access: Arithmetic, ◁┐ ARITH POLYNOMIAL

Input: Level 2/Argument 1: The dividend.
Level 1/Argument 2: The divisor.

Output: Level 2/Item 1: The quotient.
Level 1/Item 2: The remainder.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Step-by-step mode can be set (flag –100 set).
Radians mode must be set (flag –17 set).

Example: Perform the following division:

$$\frac{x^2+x+1}{2x+4}$$

Command: `DIV2(X^2+X+1,2*X+4)`

Result: `{1/2(X-1),3}`

DIV2MOD

Type: Command

Description: Performs Euclidean division on two expressions modulo the current modulus.

Access: Arithmetic, ◁┐ ARITH MODULO

Input: Level 2/Argument 1: The dividend.
Level 1/Argument 2: The divisor.

Output: Level 2/Item 1: The quotient.
Level 1/Item 2: The remainder.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
« *program* » or '*program name*'	→	

See also: HALT, NEXT

DDAYS

Type: Command

Description: Delta Days Command: Returns the number of days between two dates.
If the argument 1/level 2 date is chronologically later than the argument 2/ level 1 date, the result is negative. The range of allowable dates is October 15, 1582, to December 31, 9999.

Access: ▷┐ TIME TOOLS NXT DDAYS (TIME is the right-shift of the 9 key).
▷┐ &9 NXT DDAYS

Flags: Date Format (–42)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$date_1$	$date_2$	→	x_{days}

See also: DATE, DATE+

DEC

Type: Command

Description: Decimal Mode Command: Selects decimal base for binary integer operations. (The default base is decimal.)
Binary integers require the prefix #. Binary integers entered and returned in decimal base automatically show the suffix *d*. If the current base is not decimal, then you can enter a decimal number by ending it with d. It will be displayed in the current base when it is entered.
The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Access: ◁┐ MTH BASE DEC (MTH is the left-shift of the SYMB key).
◁┐ CONVERT BASE DEC (CONVERT is the left-shift of the 6 key).

Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12)

Input/Output: None

See also: BIN, HEX, OCT, RCWS, STWS

DECR

Type: Command

Description: Decrement Command: Takes a variable, subtracts 1, stores the new value back into the original variable, and returns the new value. The contents of *name* must be a real number or an integer.

Access: ◁┐ PRG MEMORY ARITHMETIC DECR(PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	x_{new}

Example 1: If 35.7 is stored in *A*, `'A' DECR` returns `34.7`.

Example 2: The following program counts down from 100 to 0 and leaves the integers 100 to 0 on the stack:

```
« 100 'A' STO WHILE A REPEAT 'A' DECR END 'A' PURGE »
```

See also: INCR, STO+, STO–

– If *axes* contains any strings other than 0, 1 or *n*, the DIFFEQ plotter uses the default strings 0 and 1, and plots the independent variable on the horizontal axis and the dependent variable on the vertical.

- *ptype* is a command name specifying the plot type. Executing the command DIFFEQ places the command name DIFFEQ in *PPAR*.
- *depend* is a list, { *Y y0 xErrTol* }, containing a name that specifies the dependent variable (the solution), and two numbers that specify the initial value of *Y* and the global absolute error tolerance in the solution *Y*. The default values for these elements are { *Y* 0 .0001 }

EQ must define the right-hand side of the initial value problem Y'(XF(X,Y). Y can return a real value or real vector when evaluated.

The DIFFEQ-plotter attempts to make the interval between values of the independent variable as large as possible, while keeping the computed solution within the specified error tolerance x_{ErrTol}. This tolerance may hold only at the computed points. Straight lines are drawn between computed step endpoints, and these lines may not accurately represent the actual shape of the solution. *res* limits the maximum interval size to provide higher plot resolution.

On exit from DIFFEQ plot, the first elements of *indep* and *depnd* (identifiers) contain the final values of *X* and *Y*, respectively.

If *EQ* contains a list, the initial value problem is solved and plotted using a combination of Rosenbrock (3,4) and Runge-Kutta-Fehlberg (4,5) methods. In this case DIFFEQ uses RRKSTEP to calculate y_i, and *EQ* must contain two additional elements:

- The second element of *EQ* must evaluate to the partial derivative of Y' with respect to *X*, and can return a real value or real vector when evaluated.
- The third element of *EQ* must evaluate to the partial derivative of Y' with respect to *Y*, and can return a real value or a real matrix when evaluated.

Access: ⬐ CAT DIFFEQ

Input/Output: None

See also: AXES, CONIC, FUNCTION, PARAMETRIC, POLAR, RKFSTEP, RRKSTEP, TRUTH

DIR

Type: Function

Description: Creates an empty directory structure in run mode. Can be used as an alternative to CRDIR to create an empty directory by typing DIR 'NAME' STO, which will create an empty directory 'NAME' if it does not already exist in the current directory.

Access: ⬐ CAT DIR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*DIR …END*

See also: CRDIR

DISP

Type: Command

Description: Display Command: Displays *obj* in the *n*th display line.

$n \leq 1$ indicates the top line of the display.

To facilitate the display of messages, strings are displayed without the surrounding " " delimiters. All other objects are displayed in the same form as would be used if the object were in level 1 in the multiline display format. If the object display requires more than one display line, the display starts in line *n*, and continues down the display either to the end of the object or the bottom of the display. The object displayed by DISP persists in the display only until the keyboard is ready

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'name=exp'	→	
'name(name₁ … nameₙ)=exp(name₁ … nameₙ)'	→	

Example 1: `'A=2*X' DEFINE` stores `'2*X'` in variable *A*.

Example 2: `'A(X,Y)=2*X+3/Y' DEFINE` creates a user-defined function *A*. The contents of *A* is the program `« → X Y '2*X+3/Y' »`

See also: DEF, STO, UNASSIGN

DEG

Type: Command

Description: Degrees Command: Sets Degrees angle mode.

DEG clears flags –17 and –18, and displays the DEG annunciator.

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

Access: ⬑ &MODE ANGLE DEG

⬑ PRG NXT MODES ANGLE DEG (PRG is the left-shift of the EVAL key).

Input/Output: None

See also: GRAD, RAD

DEGREE

Type: Function

Description: Returns the degree of a polynomial expression. Returns 0 for a constant expression, but –1 if the expression is zero.

Access: Catalog, ⬐ CAT

Input: Level 1/Argument 1: A polynomial expression or equation; all powers must be integers or real numbers with no fractional part.

Output: Level 1/Item 1: An integer representing the highest power in the polynomial. If the input contains powers of more than one variable, including the current variable, returns the highest power of the current variable. If the input contains powers of more than one variable, not including the current variable, returns the highest power of the first symbolic variable (one that is not stored in the current directory path). If the input contains powers of more than one variable, and all the variables are stored in the current directory path, returns the highest power of any of the variables.

Flags: If exact mode is set (flag –105 clear), the result is returned as an integer, otherwise it is returned as a real number.

Example 1: Find the degree of the polynomial represented by:
$x^2-17=x^3+2x$

Command: `DEGREE(x^2-17=x^3+2*X)`

Result: 3

DELALARM

Type: Command

Description: Delete Alarm Command: Deletes the specified alarm.
If n_{index} is 0, all alarms in the system alarm list are deleted.

Access: ⬐ TIME TOOLS ALRM DELALARM (TIME is the right-shift of the 9 key).

DIAG→

Type: Command

Description: Vector to Matrix Diagonal Command: Takes an array and a specified dimension and returns a matrix whose major diagonal elements are the elements of the array.

Real number dimensions are rounded to integers. If a single dimension is given, a square matrix is returned. If two dimensions are given, the proper order is { *number of rows, number of columns* }. No more than two dimensions can be specified.

If the main diagonal of the resulting matrix has more elements than the array, additional diagonal elements are set to zero. If the main diagonal of the resulting matrix has fewer elements than the array, extra array elements are dropped.

Access:
- ◁ MATRICES CREATE NXT DIAG→ (MATRICES is the left-shift of the 5 key).
- ◁ MTH MATRIX NXT DIAG→ (MTH is the left-shift of the SYMB key).
- ◁ MTH MATRIX MAKE NXT NXT DIAG→ (MTH is the left-shift of the SYMB key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[*array*]$_{\text{diagonals}}$	{ *dim* }	→	[[*matrix*]]

See also: →DIAG

→DIAG

Type: Command

Description: Matrix Diagonal to Array Command: Returns a vector that contains the major diagonal elements of a matrix.

The input matrix does not have to be square.

Access:
- ◁ MATRICES CREATE →DIAG (MATRICES is the left-shift of the 5 key).
- ◁ MTH MATRIX NXT →DIAG (MTH is the left-shift of the SYMB key).
- ◁ MTH MATRIX MAKE NXT NXT →DIAG (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	[*vector*]$_{\text{diagonals}}$

See also: DIAG→

DIAGMAP

Type: Command

Description: Applies a holomorphic operator to a diagonalizable matrix.

Access: Matrices, ◁ MATRICES NXT EIGENVECTORS.

Input: Level 2/Argument 1: A diagonalizable matrix.
Level 1/Argument 2: An operator, expressed as a function. The function can be stored in a variable with DEF, or can be a program, or a single expression.

Output: The matrix that results from applying the operator to the matrix.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Apply the operator e^x to the matrix $\begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{key}	→	
{ $x_{\text{key 1}}$, … ,$x_{\text{key n}}$ }	→	
0	→	
'S'	→	

See also: ASN, RCLKEYS, STOKEYS

DEPND

Type: Command

Description: Dependent Variable Command: Specifies the dependent variable (and its plotting range for TRUTH plots).

The specification for the dependent variable name and its plotting range is stored in the reserved variable *PPAR* as follows:

- If the argument is a global variable name, that name replaces the dependent variable entry in *PPAR*.
- If the argument is a list containing a global name, that name replaces the dependent variable name but leaves unchanged any existing plotting range.
- If the argument is a list containing a global name and two real numbers, or a list containing a name, array, and real number, that list replaces the dependent variable entry.
- If the argument is a list containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the dependent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is *Y*.

The plotting range for the dependent variable is meaningful only for plot type TRUTH, where it restricts the region for which the equation is tested, and for plot type DIFFEQ, where it specifies the initial solution value and absolute error tolerance.

Access: ◁ CAT DEPND

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
	'global'	→	
	{ *global* }	→	
	{ *global*, y_{start}, y_{end} }	→	
	{y_{start}, y_{end} }	→	
y_{start}	y_{end}	→	

See also: INDEP

DEPTH

Type: RPL Command

Description: Depth Command: Returns a real number representing the number of objects present on the stack (before DEPTH was executed).

Access:
- ◁ PRG STACK NXT DEPTH (PRG is the left-shift of the EVAL key).
- TOOL STACK NXT DEPTH

Input/Output:

Level n…Level 1		Level 1
	→	*n*

HERMITE

Type: Function

Description: Returns the *n*th Hermite polynomial.

Access: Arithmetic, ◁ ARITH POLY NXT

Input: A non-negative integer.

Output: The corresponding polynomial expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Find the Hermite polynomial with degree 4.

Command: `HERMITE(4)`

Result: `16*X^4-48*X^2+12`

See also: LEGENDRE, TCHEBYCHEFF

HESS

Type: Command

Description: Returns the Hessian matrix and the gradient of an expression with respect to the specified variables.

Access: Calculus ◁ CALC DERIV & INTEG

Input: Level 2/Argument 1: An expression.
Level 1/Argument 2: A vector of the variables.

Output: Level 3/Item 1: The Hessian matrix with respect to the specified variables.
Level 2/Item 2: The gradient with respect to the variables.
Level 1/Item 3: The vector of the variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Find the Hessian matrix, and the gradient with respect to each variable, of the expression:

$$t^2 + 2tu^2.$$

Command: `HESS(T^2+2*T*U^2, [T, U])`

Result: `{[[2, 2*(2*U)], [2*(2*U), 2*(2*T)]], [2*T+2*U^2, 2*T*(2*U)], [T, U]}`

See also: CURL, DIV

HEX

Type: Command

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

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.

Access: ◁ MTH BASE HEX (MTH is the left-shift of the SYMB key).
◁ CONVERT BASE HEX (CONVERT is the left-shift of the 6 key).

Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12)

Example: Find the polynomials for *u*, *v*, and *c*, where *c* is the greatest common divisor of *a* and *b* such that: $u(x^2 + 1) + v(x - 1) = c$

Command: `EGCD(X^2+1,X-1)`

Result: `{2,1,-(X+1)}`

See also: IEGCD, ABCUV

EGV

Type: Command

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

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):

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

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

Access: ◁ MATRICES NXT EIGENVECTOR EGV (MATRICES is the left-shift of the 5 key).
◁ MTH MATRIX NXT EGV (MTH is the left-shift of the SYMB key).

Input/Output:

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

See also: EGVL

EGVL

Type: Command

Description: Eigenvalues Command: Computes the eigenvalues of a square matrix.

The resulting vector L contains the computed eigenvalues.

Access: ◁ MATRICES NXT EIGENVECTOR EGVL (MATRICES is the left-shift of the 5 key).
◁ MTH MATRIX NXT EGVL (MTH is the left-shift of the SYMB key).

Input/Output:

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

See also: EGV

ELSE

Type: Command

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

See the IF and IFERR keyword entries for more information.

Access: ◁ PRG BRANCH IF ELSE (PRG is the left-shift of the EVAL key).

Input/Output: None

See also: IF, CASE, DO, ELSE, IFERR, REPEAT, THEN, UNTIL, WHILE

END

Type: Command

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

```
GROB 5 « 5 11A040A011 GXOR LASTARG GXOR »
```
turns on (makes dark) every pixel in *PICT*, then superimposes a 5 x 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.

See also: GOR, REPL, SUB

*H

Type: Command

Description: Multiply Height Command: Multiplies the vertical plot scale by x_{factor}. *H is provided for compatibility with the HP 48. *H is the same as SCALEH; see its listing for details.

HADAMARD

Type: Command

Description: Performs an element by element multiplication of two matrices (Hadamard product).

Access: Matrices, ◀ MATRICES OPERATIONS NXT

Input: Level 2/Argument 1: Matrix 1.
Level 1/Argument 2: Matrix 2.
The matrices must have the same order.

Output: The matrix representing the result of the multiplication.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Find the Hadamard product of the following two matrices:

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

Command: `HADAMARD([[3,-1,2][0,1,4]],[2,3,0][1,5,2]])`

Result: `[[6,-3,0][0,5,8]]`

HALFTAN

Type: Command

Description: Transforms an expression by replacing sin(x), cos(x) and tan(x) subexpressions with tan(x/2) terms.

Access: Trigonometry, ⌐ TRIG or SYMB TRIG

Input: An expression

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

See also: TAN2CS2, TAN2SC2

HALT

Type: Command

Description: Halt Program Command: Halts program execution.

Program execution is halted at the location of the HALT command in the program. The HLT annunciator is turned on. Program execution is resumed by executing CONT (that is, by pressing ◀ CONT). Executing KILL cancels all halted programs.

Example: Replace with zero the terms smaller than EPS in the expression: $10^{-13}x + 10^{-2}$

Command: `EPSX0(1E-13*X+.01)`

Result: `0*X+.01`

EQNLIB

Type: Command

Description: Starts the Equation Library application.

Access: APPS EQUATION LIBRARY

Input/Output: None

See also: MSOLVR, SOLVEQN

EQW

Type: Command

Description: Opens Equation Writer, where you can edit an expression.

Puts an object into the Equation Writer.

Access: ⌐ CAT EQW

(Non-programmable access is via ▼ when there is an algebraic object on the stack. To start a new equation when not entering a program object, press ⌐ EQW)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
exp_1	→	exp_2

See also: EDIT, EDITB, VISIT, VISITB

EQ→

Type: Command

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

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

Access: ◀ PRG TYPE NXT EQ→ (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
'$symb_1$=$symb_2$'	→	'$symb_1$'	'$symb_2$'
z	→	z	*0*
'*name*'	→	'*name*'	*0*
'*x_unit*'	→	'*x_unit*'	*0*
'*symb*'	→	'*symb*'	*0*

See also: ARRY→, DTAG, LIST→, OBJ→, STR→

ERASE

Type: Command

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

Access: ⌐ CAT ERASE

Input/Output: None

See also: DRAW

Result: `-1`

Note this is the remainder of the input polynomial modulo the term *x* in the Grœbner basis

See also: GBASIS

GRIDMAP

Type: Command

Description: GRIDMAP Plot Type Command: Sets the plot type to GRIDMAP.

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 that 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:

$$\{ (x_{min}, y_{min}), (x_{max}, y_{max}), \text{indep, res, axes, ptype, depth} \}$$

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*.

Access: [→] [CAT] GRIDMAP

Input/Output: None

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

→GROB

Type: Command

Description: Stack to Graphics Object Command: Creates a graphics object from a specified object, where the argument $n_{char\ size}$ specifies the character size of the object.

$n_{char\ size}$ 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 Equation Writer application picture.

Access: [→] [CAT] →GROB

[←] [PRG] [NXT] GROB →GROB ([PRG] is the left-shift of the [EVAL] key).

EVAL

Type: Command

Description: Evaluate Object Command: Evaluates the object.

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

Object 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.
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 Results mode (flag –3 set) or execute →NUM on that argument.

Access: [EVAL]

Flags: Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	*(see above)*

See also: →NUM, SYSEVAL

EXLR

Type: Command

Description: Returns the left- and right-hand sides of an equation as discrete expressions.

Input/Output:

L_2/A_1	L_1/A_2		L_3/I_1	L_2/I_2	L_1/I_3
[[matrix]]	$n_{position1}$	→	*[[matrix]]*	$n_{position2}$	$\tilde{x}_{get}$
[[matrix]]	$\{ n_{row}, m_{column} \}_1$	→	*[[matrix]]*	$\{ n_{row}, m_{column} \}_2$	$\tilde{x}_{get}$
'$name_{matrix}$'	$n_{position1}$	→	'$name_{matrix}$'	$n_{position2}$	$\tilde{x}_{get}$
'$name_{matrix}$'	$\{ n_{row}, m_{column} \}_1$	→	'$name_{matrix}$'	$\{ n_{row}, m_{column} \}_2$	$\tilde{x}_{get}$
[vector]	$n_{position}$	→	*[vector]*	$n_{position2}$	$\tilde{x}_{get}$
[vector]	$\{n_{position1}\}$	→	*[vector]*	$\{n_{position2}\}$	$\tilde{x}_{get}$
'$name_{vector}$'	$n_{position1}$	→	'$name_{vector}$	$n_{position2}$	$\tilde{x}_{get}$
'$name_{vector}$'	$\{n_{position1}\}$	→	'$name_{vector}$	$\{n_{position2}\}$	$\tilde{x}_{get}$
{ *list* }	$n_{position1}$	→	{ *list* }	$n_{position2}$	obj_{get}
{ *list* }	$\{n_{position1}\}$	→	{ *list* }	$\{n_{position2}\}$	obj_{get}
'$name_{list}$'	$n_{position1}$	→	'$name_{list}$'	$n_{position2}$	obj_{get}
'$name_{list}$'	$\{n_{position1}\}$	→	'$name_{list}$'	$\{n_{position2}\}$	obj_{get}

L = Level; A = Argument; I = Item

See also: GET, PUT, PUTI

GOR

Type: Command

Description: 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*.
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 first argument (stack level 3) is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the first argument (level 3) is *PICT*, no result is returned to the stack.
Any portion of $grob_1$ that extends past $grob_{target}$ or *PICT* is truncated.

Access: ← PRG NXT GROB GOR (PRG is the left-shift of the EVAL key).

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 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)	$grob_1$	→	

See also: GXOR, REPL, SUB

GRAD

Type: Command

Description: Grads Mode Command: Sets Grads angle mode.
GRAD clears flag –17 and sets flag –18, and displays the GRD 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.

Access: ← & MODE ANGLE GRAD
→ CAT GRAD

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\tilde{x}$	→	$e^{\tilde{x}}$
'*symb*'	→	'*EXP(symb)*'

See also: ALOG, EXPM, LN, LOG

EXP2HYP

Type: Function

Description: Converts expressions involving the exponential function into expressions with hyperbolic functions.

Access: Catalog, → CAT

Input: An expression

Output: The rewritten expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Rewrite in terms of hyperbolic functions the expression $e^{5 \cdot \ln(x)}$

Command: `EXP2HYP(EXP(5*LN(X)))`

Result: `SINH(5*LN(X))+COSH(5*LN(X))`

EXP2POW

Type: Function

Description: Simplifies expressions involving the composition of the exponential and logarithmic functions. Compare this to LNCOLLECT which combines logarithmic terms; the difference is shown in the results of the second example used here and for LNCOLLECT.

Access: ← CONVERT REWRITE

Input: An expression

Output: The simplified expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example 1: Simplify the expression $e^{5 \cdot \ln(x)}$

Command: `EXP2POW(EXP(5*LN(X)))`

Result: x^5

Example 2: Simplify the expression $e^{n \cdot \ln(x)}$

Command: `EXP2POW(EXP(N*LN(X)))`

Result: x^N

See also: LNCOLLECT

EXPAN

Type: Command

Description: Expand Products Command: Rewrites an algebraic expression or equation by expanding products and powers. This command is similar to the old HP 48G series command, with minor modifications (such as adding RISCH for integration).

Access: → CAT EXPAN

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Find a Grœbner basis of the ideal polynomial generated by the polynomials:
$x^2 + 2xy^2$, $xy + 2y^3 - 1$

Command: `GBASIS([X^2 + 2*X*Y^2, X*Y + 2*Y^3 - 1], [X,Y])`

Result: `[X, 2*Y^3-1]`
Note this is not the *minimal* Grœbner basis, as the leading coefficient of the second term is not 1; the algorithm used avoids giving results with fractions.

See also: GREDUCE

GCD

Type: Function

Description: Returns the greatest common divisor of two objects.

Access: Arithmetic, ◁ ARITH POLY NXT

Input: Level 2/Argument 1: An expression, or an object that evaluates to a number.
Level 1/Argument 2: An expression, or an object that evaluates to a number.

Output: The greatest common divisor of the two objects.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Find the greatest common divisor of 2805 and 99.

Command: `GCD(2805,99)`

Result: `33`

See also: GCDMOD, EGCD, IEGCD, LCM

GCDMOD

Type: Function

Description: Finds the greatest common divisor of two polynomials modulo the current modulus.

Access: Arithmetic, ◁ ARITH MODULO

Input: Level 2/Argument 1: A polynomial expression.
Level 1/Argument 2: A polynomial expression.

Output: The greatest common divisor of the two expressions modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Find the greatest common divisor of 2x^2+5 and 4x^2-5x, modulo 13.

Command: `GCDMOD(2X^2+5,4X^2-5X)`

Result: `-(4X-5)`

See also: GCD

GET

Type: Command

Description: Get Element Command: Returns from the argument 1/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 argument 2/level 1. For matrices, $n_{position}$ is incremented in *row* order.

EXPFIT

Type: Command

Description: 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.
LINFIT is the default specification in *ΣPAR*.

Access: ◁ CAT EXPFIT

Input/Output: None

See also: BESTFIT, LR, LINFIT, LOGFIT, PWRFIT

EXPLN

Type: Command

Description: Transforms the trigonometric terms in an expression to exponential and logarithmic terms.

Access: ◁ EXP&LN or Convert, ◁ CONVERT REWRITE or SYMB NXT EXP & LN

Input: An expression

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
Complex mode must be set (flag –103 set).

Example: Transform the following expression and simplify the result using the EXPAND command:
$2\cos(x^2)$

Command: `EXPLN(2*COS(X^2))`
`EXPAND(ANS(1))`

Result: `(EXP(i*X^2)^2+1)/EXP(i*X^2)`

See also: SINCOS

EXPM

Type: Analytic Function

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

Access: ◁ MTH HYPERBOLIC NXT EXPM (MTH is the left-shift of the SYMB key).
◁ EXP&LN EXPM (EXP&LN is the left-shift of the 8 key).

Flags: Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	$e^x - 1$
'symb'	→	*'EXPM(symb)'*

See also: EXP, LNP1

EYEPT

Type: Command

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

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.

Access: ▶ CAT FUNCTION
Flags: Simultaneous Plotting (–28), Curve Filling (–31)
Input/Output: None
See also: BAR, CONIC, DIFFEQ, FAST3D, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

FXND

Type: Command
Description: Splits an object into a numerator and a denominator.
Access: Catalog, ▶ CAT
Input: A fraction, or an object that evaluates to a fraction.
Output: The object split into numerator and denominator.
Level 2/Item 1: The numerator.
Level 1/Item 2: The denominator.
Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Example: Return the numerator and the denominator of the following expression:

$$\frac{(x-3)^2}{z+4}$$

Command: `FXND((X-3)^2/(Z+4))`
Result: `{(X-3)^2, Z+4}`
See also: EXLR

Input: An expression or an integer.
Output: The factorized expression, or the integer expressed as the product of prime numbers.
Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Results including complex terms are returned if complex mode is set (flag –103 set).
Example: Factorize the following:
$x^2 + 5x + 6$
Command: `FACTOR(X^2+5*X+6)`
Result: `(X+2)(X+3)`
See also: EXPAN, EXPAND

FACTORMOD

Type: Function
Description: Factorizes a polynomial modulo the current modulus. The modulus must be less than 100, and a prime number, it can be changed by MODSTO.
Access: Arithmetic, ◀ ARITH MODULO
Input: The expression to be factorized.
Output: The factorized expression modulo the current modulus.
Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
Example: Factorize the following expression modulo 3.
x^2+2
Command: `FACTORMOD(X^2+2)`
Result: `(X+1)*(X-1)`
See also: MODSTO

FACTORS

Type: Command
Description: For a value or expression, returns a list of prime factors and their multiplicities.
Access: Arithmetic, ◀ ARITH
Input: A value or expression.
Output: A list of prime factors of the value or expression, with each factor followed by its multiplicity expressed as a real number.
Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
Example 1: Find the prime factors of 100.
Command: `FACTORS(100)`
Result: `{5 2. 2 2.}`
Example 2: Find the irreducible factors of: $x^2 + 4x + 4$
Command: `FACTORS(X^2+4*X+4)`
Result: `{X+2,2.}`

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

So, for example, 2 FREEZE freezes the history/stack/command-line area, 3 FREEZE freezes the status area and the history/stack/command-line area, and 7 FREEZE freezes all three areas. Values of $n_{\text{display area}} \geq 7$ 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).

Access: ◀ PRG NXT OUT FREEZE (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{displayarea}}$	→	

Example 1: 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.

Example 2: This program:

```
« ( # 0d # 0d ) 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.

Flags: None

See also: CLLCD, DISP, HALT

FROOTS

Type: Command

Description: For a rational polynomial, returns an array of its roots and poles, with their corresponding multiplicities. This is the inverse of FCOEF and uses the same notation for roots and poles.

Access: Arithmetic, ◀ ARITH POLY NXT

Input: A rational polynomial.

Output: An array of the form [Root 1, Multiplicity 1, Root 2, Multiplicity 2, . . .]
A negative multiplicity indicates a pole.

Flags: Exact mode must be set (flag –105 clear). Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
If complex mode is set (flag –103 set), FROOTS looks for complex solutions as well as real solutions.

If approximate mode is set (flag –105 set) FROOTS searches for numeric roots.

See also: FCOEF

FS?

Type: Command

Description: 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.

Access: ◀ PRG TEST NXT NXT FS? (PRG is the left-shift of the EVAL key).

Access: ◀ CAT FAST3D

Input/Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

FCOEF

Type: Command

Description: From an array of roots and multiplicities/poles, returns a rational polynomial with a leading coefficient of 1, with the specified set of roots or poles, and with the specified multiplicities.

Access: Arithmetic, ◀ ARITH POLY NXT

Input: An array of the form [Root 1, multiplicity/pole 1, Root 2, multiplicity/pole 2, . . .] The multiplicity/pole must be an integer. A positive number signifies a multiplicity. A negative number signifies a pole.

Output: The rational polynomial with the specified roots and multiplicities/poles. The polynomial is written using the current independent variable.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Find the rational polynomial corresponding to the following set of roots and poles:
1, 2, 3, –1

Command: `FCOEF([1,2,3,-1])`

Result: `(X-1)^2/(X-3)`

See also: FROOTS

FC?

Type: Command

Description: 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.

Access: ◀ PRG TEST NXT NXT FC? (PRG is the left-shift of the EVAL key).
◀ PRG NXT MODES FLAG FC? (PRG is the left-shift of the EVAL key).
◀ &MODE FLAG FC?

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?C, FS? FS?C, SF

FC?C

Type: Command

Description: 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.

Access: ◀ PRG TEST NXT NXT FC?C (PRG is the left-shift of the EVAL key).
◀ PRG NXT MODES FLAG FC?C (PRG is the left-shift of the EVAL key).
◀ &MODE FLAG FC?C

FOR

Type: Command Operation

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

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 RPL syntax is this:
  x_{start} x_{finish} FOR counter loop-clause NEXT
  The algebraic syntax is this:
  FOR(counter, x_{start}, x_{finish}) loop-clause NEXT
  FOR takes x_{start} and x_{finish} 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 RPL is:
  x_{start} x_{finish} FOR counter loop-clause $x_{increment}$ STEP
  The algebraic syntax is:
  FOR(counter, x_{start}, x_{finish}) loop-clause, STEP ($x_{increment}$)
  FOR takes x_{start} and x_{finish} as the beginning and ending values for the loop counter, then creates the local variable counter as a loop counter. Next, the loop clause is executed; counter can be referenced or have its value changed within the loop clause. STEP takes $x_{increment}$ 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.

Access: ◁ PRG BRANCH FOR (PRG is the left-shift of the EVAL key).

Input/Output:

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

Note: It should be noted that FOR inputs may also be integers (object type 28) and binary integers (type 10). FOR actually runs fastest on binary integers, runs "normally" on reals and slightly slower on integers.

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

```
« 0 1 100 FOR I I + 2 STEP »
```

See also: NEXT, START, STEP

Access: ◁ MTH (NXT) FFT FFT (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[array]$_1$	→	*[array]*$_2$

See also: IFFT

FILER

Type: Command

Description: Opens File Manager.

Access: ◁ FILES (FILES is the left-shift of the APPS key).
▷ CAT FILER

Input/Output: None

FINDALARM

Type: Command

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

If the input is a real number *date*, FINDALARM returns the index of the first alarm due after 12:00 AM on that date. If the input is a list { *date time* }, it returns the index of the first alarm due after that date and time. If the input 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.

Access: ▷ TIME TOOLS ALRM FINDALARM (TIME is the right-shift of the 9 key).
▷ & 9 ALRM FINDALARM
◁ PRG (NXT) (NXT) TIME ALRM FINDALARM (PRG is the left-shift of the EVAL key).

Flags: Date Format (–42)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
date	→	n_{index}
{ *date time* }	→	n_{index}
0	→	n_{index}

See also: DELALARM, RCLALARM, STOALARM

FINISH

Type: Command

Description: Finish Server Mode Command: Terminates Kermit Server mode in a device connected to the calculator.

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.

Access: ▷ CAT FINISH

Flags: I/O Device flag (–33), I/O Messages (–39), I/O Device for Wire (–78)

Input/Output: None

See also: BAUD, CKSM, KGET, PARITY, PKT, RECN, RECV, SEND, SERVER

FIX

Type: Command

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

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\tilde{z}$	→	$ln\ \tilde{z}$
'*symb*'	→	'*LN(symb)*'

See also: ALOG, EXP, ISOL, LNP1, LOG

LNAME

Type: Command

Description: Returns the variable names contained in a symbolic expression.

Access: Catalog, ▶ CAT

Input: A symbolic expression.

Output: Level 2/Argument 1: The original expression.
Level 1/Argument 2: A vector containing the variable names. The variable names are sorted by length, longest first, and ones of equal length are sorted alphabetically.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: List the variables in the expression COS(B)/2*A + MYFUNC(PQ) + 1/T.

Command:
```
LNAME(COS(B)/2*A + MYFUNC(PQ) + INV(T))
```

Result:
```
{COS(B)/2*A + MYFUNC(PQ) + 1/T, [MYFUNC,PQ,A,B,T]}
```

See also: LVAR

LNCOLLECT

Type: Command

Description: Simplifies an expression by collecting logarithmic terms. For symbolic powers does not perform the same simplification as EXP2POW; compare example 2 here with example 2 for EXP2POW.

Access: Algebra, ▶ ALG , ◀ EXP&LN, or SYMB NXT EXP & LN, or ◀ CONVERT REWRITE NXT

Input: An expression.

Output: The simplified expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example 1: Simplify the following expression:
$2(\ln(x)+\ln(y))$

Command:
```
LNCOLLECT(2(LN(X)+LN(Y))
```

Result:
```
LN(X^2*Y)
```

Example 2: Compare the effect of LNCOLLECT with the effect of EXP2POW on the expression $e^{n\cdot\ln(x)}$

Command:
```
LNCOLLECT(EXP(N*LN(X))
```

Result:
```
EXP(N*LN(X))
```

See also: EXP2POW, TEXPAND

LNP1

Type: Analytic function

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

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.

Access: ◀ MTH HYPERBOLIC LNP1 (MTH is the left-shift of the SYMB key).

Input/Output: None

See also: BIN, DEC, OCT, RCWS, STWS

HILBERT

Type: Command

Description: Returns a square Hilbert matrix of the specified order.

Access: Matrices, ◀ MATRICES CREATE NXT or ◀ MTH MATRX MAKE NXT NXT

Input: A positive integer, representing the order.

Output: The Hilbert matrix of the specified order.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Find the order 3 Hilbert matrix.

Command:
```
HILBERT(3)
```

Result:

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

See also: CON, IDN, RANM, VANDERMONDE

HISTOGRAM

Type: Command

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

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:

{ (xmin, ymin) (xmax, ymax) indep res axes ptype depend }

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) for the HP 48gII and (–6.5,–3.9) for the HP 50g and 49g+.
- (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) for the HP 48gII and (6.5,4.0) for the HP 50g and 49g+.
- *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*.

Input/Output:

Level$_{n+1}$/Argument$_1$ …Level$_2$/Argument$_n$	Level$_1$/Argument$_{n+1}$		Level 1/Item 1
obj$_1$ … *obj*$_n$	*n*	→	{ *obj*$_1$, … ,*obj*$_n$ }

Example: The program « DEPTH →LIST 'A' STO » combines the entire contents of the stack into a list that is stored in variable *A*.

See also: →ARRY, LIST→, →STR, →TAG, →UNIT

ΔLIST

Type: Command

Description: List Differences Command: Returns the first differences of the elements in a list. Adjacent elements in the list must be suitable for mutual subtraction.

Access: ◀ MTH LIST ΔLIST (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ *list* }	→	{ *differences* }

Example 1: { 4 20 1 17 60 91 } ⏢LIST returns { 16 -19 16 43 31 }.

Example 2: { A B C 1 2 3 } ⏢LIST returns { 'B-A' 'C-B' '1-C' 1 1 }.

Example 3: { 'A+3' 'X/5' 'Y^4' } ⏢LIST returns { 'X/5-(A+3)' 'Y^4-X/5' }.

See also: ΣLIST, ΠLIST, STREAM

ΠLIST

Type: Command

Description: List Product Command: Returns the product of the elements in a list. The elements in the list must be suitable for mutual multiplication.

Access: ◀ MTH LIST ΠLIST (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ *list* }	→	*product*

Example 1: { 5 8 2 } ΠLIST returns 80.

Example 2: { A B C 1 } ΠLIST returns 'A*B*C'.

See also: ΣLIST, ΔLIST, STREAM

ΣLIST

Type: Command

Description: List Sum Command: Returns the sum of the elements in a list. The elements in the list must be suitable for mutual addition.

Access: ◀ MTH LIST ΣLIST (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ *list* }	→	*sum*

Example 1: { 5 8 2 } ΣLIST returns 15.

Example 2: { A B C 1 } ΣLIST returns 'A+B+C+1'.

See also: ΠLIST, ΔLIST, STREAM

LN

Type: Analytic function

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

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 (hours or degrees).
- *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.

Access: ◀ TIME Tools NXT HMS+ (TIME is the right-shift of the 9 key).

◀ & 9 NXT HMS+

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
HMS$_1$	*HMS*$_2$	→	*HMS*$_1$ + *HMS*$_2$

See also: HMS→, →HMS, HMS–

HMS→

Type: Command

Description: 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).

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 (hours or degrees).
- *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.

Access: ◀ TIME Tools NXT HMS→ (TIME is the right-shift of the 9 key).

◀ & 9 NXT HMS→

Input/Output:

Level 1/Argument 1		Level 1/Item 1
HMS	→	*x*

See also: →HMS, HMS+, HMS–

→HMS

Type: Command

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

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.

Access: ◀ TIME Tools NXT →HMS (TIME is the right-shift of the 9 key).

◀ & 9 NXT →HMS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*HMS*

See also: HMS→, HMS+, HMS–

Example: Linearize the following expression:
$x(e^x e^y)^4$

Command: `LIN(X*(EXP(X)*EXP(Y))^4)`

Result: `X*EXP(4X+4Y)`

See also: TEXPAND

LINE

Type: Command Operation

Description: Draw Line Command: Draws a line in *PICT* between the input coordinates.

Access: ◀ PRG NXT PICT LINE (PRG is the left-shift of the EVAL key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
(x_1, y_1)	(x_2, y_2)	→	
$\{ \#n_1, \#m_1 \}$	$\{ \#n_2, \#m_2 \}$	→	

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.

See also: ARC, BOX, TLINE

ΣLINE

Type: Command

Description: 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*.
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

Access: ▶ CAT ΣLINE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	'*symb*$_{\text{formula}}$'

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

See also: BESTFIT, COLΣ, CORR, COV, EXPFIT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, XCOL, YCOL

LINFIT

Type: Command

Description: 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.
LINFIT is the default specification in *ΣPAR*.

Access: ▶ CAT LINFIT

Output: Level 2/Item 1: The value for *u*.
Level 1/Item 2: The value for *v*.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Find a solution in integers of the equation:
$6a + 11b = 3$

Command: `IABCUV(6,11,3)`

Result: `{6,-3}`

See also: ABCUV, IEGCD

IBASIS

Type: Command

Description: Determines the basis of the intersection between two vector spaces.

Access: Matrices, ◀ MATRICES NXT VECTOR

Input: Two lists of vectors

Output: A list of vectors.

Flags: Exact mode must be set (flag –105 clear).

Example: Find a vector of a basis of the intersection of the vector sub-spaces defined by [1, 2] and [2, 4]

Command: `IBASIS({[1,2]}, {[2,4]})`

Result: `{[1,2]}`

See also: BASIS

IBERNOULLI

Type: Function

Description: Returns the *n*th Bernoulli number for a given integer *n*.

Access: Arithmetic, ◀ ARITH INTEGER

Input: Level 1/Argument 1: an integer.

Output: Level 1/Item 1: The corresponding *n*th Bernoulli number for the integer. For numbers greater than about 40 the calculation can take a long time.

Flags: Numeric mode must not be set (flag –3 clear).

IBP

Type: Command

Description: Performs integration by parts on a function. The function must be able to be represented as a product of two functions, where the antiderivative of one of the functions is known:
`f(x) = u(x).v'(x)`

Note that the command is designed for use in RPN mode only.

Access: SYMB CALC or Calculus, ◀ CALC DERIV & INTEG NXT

Input: Level 2: The integrand expressed as a product of two functions, `u(x).v'(x)`
Level 1: The antiderivative of one of the component functions, v(x).

Output: Level 2: u(*x*)v(*x*)
Level 1: -u'(*x*)v(*x*)

Result: `COS(X)*(cC0 -X)+(cC1 – -1)*SIN(X)`

See also: DESOLVE

LEGENDRE

Type: Function

Description: Returns the *n*th degree Legendre polynomial.

Access: Arithmetic, ◁ ARITH POLYNOMIAL NXT NXT

Input: An integer, *n*.

Output: The *n*th Legendre polynomial.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Find the Legendre polynomial with degree 4.

Command: `LEGENDRE(4)`

Result: `(35*X^4-30*X^2+3)/8`

See also: HERMITE, TCHEBYCHEFF

LGCD

Type: Function

Description: Returns the greatest common divisor of a list of expressions or values.

Access: Arithmetic, ◁ ARITH NXT

Input: A list of expressions or values.

Output: Level 2/Item 1: The list of elements.
Level 1/Item 2: The greatest common divisor of the elements.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

See also: GCD

LIBEVAL

Type: Command

Description: Evaluate Library Function Command: Evaluates unnamed library functions.

> **WARNING: Use extreme care when executing this function. Using LIBEVAL with random addresses will almost always cause a memory loss. Do not use this function unless you know what you are doing.**

#n$_{function}$ is of the form *lllfff*h, where *lll* is the library number, and *fff* the function number.

Access: ▷ CAT LIBEVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#$n_{function}$	→	

See also: EVAL, FLASHEVAL, SYSEVAL

LIBS

Type: Command

Description: Libraries Command: Lists the title, number, and port of each library attached to the current directory. The title of a library often takes the form *LIBRARY-NAME* : *Description*. A library without a title is displayed as " ".

- 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).

Access: ◁ MATRICES CREATE IDN (MATRICES is the left-shift of the 5 key).
◁ MTH MATRIX MAKE IDN (MTH is the left-shift of the SYMB key).

Input/Output:

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

See also: CON

IDIV2

Type: Command

Description: For two integers, *a* and *b*, returns the integer part of *a*/*b*, and the remainder, *r*.

Access: Arithmetic, ◁ ARITH INTEGER

Input: Level 2/Argument 1: *a*.
Level 1/Argument 2: *b*.

Output: Level 2/Item 1: The integer part of *a*/*b*.
Level 1/Item 2: The remainder.

Flags: Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Return the integer part and the remainder of 11632/864.

Command: `IDIV2(11632,864)`

Result: `{13,400}`

See also: DIV2, IQUOT

IEGCD

Type: Command

Description: Given two integers *x* and *y*, returns three integers, *a*, *b*, and *c*, such that:
$ax+by=c$
where *c* is the GCD of *x* and *y*.

Access: SYMB ARITH or Arithmetic, ◁ ARITH INTEGER

Input: Level 2/Argument 1: *x*.
Level 1/Argument 2: *y*.

Output: Level 3/Item 1: *c*.
Level 2/Item 2: *a*.
Level 1/Item 3: *b*.
Note the order, *c* is first.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Find *a*, *b* and *c* such that *a*18 + *b*24 = *c*, where *c* is the GCD of 18 and 24.

Command: `IEGCD(18,24)`

Result: `{6,-1,1}`

LASTARG

Type: Command

Description: Returns copies of the arguments of the most recently executed command. 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 program, the last arguments saved are from the evaluated algebraic expression or program, not from the original command.

Access: ◁ CAT LASTARG

◁ PRG NXT NXT ERROR LASTA (PRG is the left-shift of the EVAL key).

◁ ANS in RPN mode. (ANS is the left-shift of the ENTER key).

Flags: Last Arguments (–55)

Input/Output:

Level 1		Level n	…	Level 1
	→	*obj*$_n$	…	*obj*$_1$

See also: ANS, LAST

LCD→

Type: Command

Description: LCD to Graphics Object Command: Returns the current stack and menu display as a 131 × 80 (on the HP 50g and 49g+) or 131 × 64 (on the HP 48gII) graphics object.

Access: ◁ PRG NXT GROB NXT LCD→ (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*grob*

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.

See also: →GROB, →LCD

→LCD

Type: Command

Description: Graphics Object to LCD Command: Displays the specified graphics object with its upper left pixel in the upper left corner of the display. If the graphics object is larger than 131 × 72 (on the HP 50g and 49g+) or 131 × 56 (on the HP 48gII), it is truncated.

Access: ◁ PRG NXT GROB NXT →LCD (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
grob	→	

See also: BLANK, →GROB, LCD→

LCM

Type: Function

Description: Returns the least common multiple of two objects.

Access: Arithmetic, ◁ ARITH POLYNOMIAL NXT NXT

Input: Level 2/Argument 1: An expression, a number, or object that evaluates to a number.
Level 1/Argument 2: An expression, a number, or object that evaluates to a number.

3 The key buffer is cleared.

4 If any or all of the display is "frozen" (by FREEZE), that state is cancelled.

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.

Access: ◁ PRG NXT NXT ERROR [IFERR] IFERR (PRG is the left-shift of the EVAL key).

Flags: Last Arguments (–55)

Input/Output: None

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 » »
```

See also: CASE, ELSE, END, IF, THEN

IFFT

Type: Command

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

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

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

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

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

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

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

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.

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.

Access: [▶] CAT LABEL
Input/Output: None
See also: AXES, DRAW, DRAX

LAGRANGE

Type: Command
Description: Returns the interpolating polynomial of minimum degree for a set of pairs of values. For two pairs, DROITE will fit a straight line.
Access: Arithmetic, [◀] ARITH POLY [NXT]
Input: A two × *n* matrix of the *n* pairs of values.
Output: The polynomial that results from the Lagrange interpolation of the data.
Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
Example: Find an interpolating polynomial for the data (1,6), (3,7), (4,8), (2,9)
Command: `LAGRANGE([[1,3,4,2][6,7,8,9]])`
Result: $\frac{8x^3 - 63x^2 + 151x - 60}{6}$
See also: DROITE

LANGUAGE→

Type: Command
Description: Language: Returns the language that is currently set. 0 for English, 1 for French, and 2 for Spanish.
Access: [▶] CAT LANGUAGE→
Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*value*

See also: →LANGUAGE

→LANGUAGE

Type: Command
Description: Language: Sets the language for things such as error messages: 0 for English, 1 for French, and 2 for Spanish.
Access: [▶] CAT →LANGUAGE
Input/Output:

Level 1/Argument 1		Level 1/Item 1
value	→	

See also: LANGUAGE→

Access: Calculus, [◀] CALC DIFFERENTIAL EQNS
Input: A rational expression.
Output: The inverse Laplace transformation of the expression.
Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
Example: Find the inverse Laplace transform of $\frac{1}{(x-5)^2}$
Command: `ILAP(1/(X-5)^2)`
Result: `X*EXP(5*X)`
See also: LAP, LAPL

IM

Type: Function
Description: Imaginary Part Function: Returns the imaginary part of its complex argument.
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.
Access: [▶] CMPLX IM (CMPLX is the right-shift of the [I] key).
Flags: Numerical Results (–3)
Input/Output:

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

See also: C→R, RE, R→C

IMAGE

Type: Command
Description: Computes the basis of the image (also called the range) of a linear application *f*.
Access: Matrices, [◀] MATRICES LINEAR APPL
Input: A matrix representing a linear application *f* in terms of the standard basis.
Output: A list of vectors representing a basis of the image of *f*.
Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Example: Find the image of $\begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 3 & 1 & 4 \end{bmatrix}$
Command: `IMAGE([1,1,2] [2,1,3] [3,1,4])`
Result: `{[1,0,-1] [0,1,2]}`
See also: BASIS, KER

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 ALPHA.

Access: ◀ PRG NXT IN KEY (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
	→	$x_{n\,m}$	*1*
	→		*0*

Example: The program « DO UNTIL KEY END 81 SAME » returns 1 to the stack if the ◀ key is pressed while the indefinite loop is running.

See also: WAIT, KEYEVAL

KEYEVAL

Type: Command

Description: Actions the specified key press.

You input a number, in the format ***ab.c***, that represents the key. In the number ***ab.c:***

- ***a*** is the row coordinate number, where row 1 is the top-most row.
- ***b*** is the column number, where column 1 is the left-most column.
- ***c*** is the shift state of the key, i.e., whether it is normal, alpha-shifted, left shifted, etc.

The shift state representations are as follows:

1: Normal function.	
2: Left-shift function.	21: Left shift-and-hold function.
3. Right-shift function.	31: Right shift-and-hold function.
4. Alpha-function.	41: Alpha shift-and-hold function.
5. Alpha-left-shift function.	51: Alpha-left-shift-and-hold function.
6. Alpha-right-shift function.	61: Alpha-right-shift-and-hold function.

The sign of the input controls whether USER mode key assignments are used. Positive inputs specify the USER mode key definition. Negative inputs specify the default system keyboard.

Access: ▶ CAT KEYEVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
nn.n	→	

Example: Turn the calculator off using a command.

Command: `KEYEVAL(101.3)`

Result: The calculator is turned off.

→KEYTIME

Type: Command

Description: Sets a new keytime value.

Keytime is the time after a keypress during which further keypresses will not be actioned. It is measured in ticks, with valid values between 0 and 4096 ticks. If you experience key bounce, you can increase the value of keytime. If you experience lost keystrokes when rapidly hitting the same key in succession, you can decrease the value of keytime. The default is 1138 ticks.

Access: ▶ CAT →KEYTIME

Input/Output:

Level 1/Argument 1		Level 1/Item 1
time	→	

See also: KEYTIME→

Variable	Function
{ 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 information about object types, see the TYPE command. When creating a multi-column dialog box, you can span columns by using an empty list as a field definition. A field that appears to the left of an empty field automatically expands to fill the empty space.
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* } in item 1 or level 2, and puts a 1 in item 2 or level 1. (If a field is empty, NOVAL is returned as a place holder.) If you exit the dialog box by selecting CANCEL or F2, INFORM returns 0.

Access: ◀ PRG NXT IN INFORM (PRG is the left-shift of the EVAL key).

Input/Output:

L_5/A_1	L_4/A_2	L_3A_3	L_2/A_4	L_1/A_5		L_2/I_1	L_1/I_2
"title"	{ s_1 s_2 … s_n }	*format*	{ *resets* }	{ *init* }	→	{ *vals* }	*1*
"title"	{ s_1 s_2 … s_n }	*format*	{ *resets* }	{ *init* }	→		*0*

L = Level; A = Argument; I = item

Example: Place the following five lines on the stack and run INFORM:

```
"The Title"
{ { "ONE" "Name?" 2 } { } { "TWO" "Age?" }
  { "THREE" "Lucky numbers?" 5 } }
{ 2 }
{ NOVAL NOVAL { 1 2 3 } }
{ "Charlotte" NOVAL { 4 5 6 } }
```

See also: CHOOSE, INPUT, NOVAL, TYPE

INPUT

Type: Command

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

When INPUT is executed, the stack or history 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 screen. Depending on the second argument (level 1), 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.

Example 1: Analyze the isometry given by the matrix $\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$

Command: `ISOM([[0,-1] [-1,0]])`

Result: { [1, 1] –1}, meaning the matrix represents a symmetry in the line $y = -x$, and this is an indirect isometry.

Example 2: Analyze the isometry given by the matrix $\begin{bmatrix} \frac{1}{2} & \frac{-\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}$

Command: `ISOM([[1/2, -√3/2][√3/2, 1/2]])`

Result: { π/3, 1 }, meaning the matrix represents a rotation of π/3 radians, and this is a direct isometry.

See also: MKISOM

ISPRIME?

Type: Function

Description: Tests if a number is prime. For numbers of the order of 10^{14} or greater (to be exact, greater than 341550071728321), tests if the number is a pseudoprime; this has a chance of less than 1 in 10^{12} of wrongly identifying a number as a prime.

Access: SYMB ARITH or Arithmetic, ◁ ARITH INTEGER NXT

Input: An object that evaluates to an integer or a whole real number.

Output: 1 (True) if the number is prime, 0 (False) if it is not.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

See also: NEXTPRIME, PREVPRIME

I→R

Type: Function

Description: Converts an integer into a real number.

Access: ▷ CONVERT REWRITE

Flags: Exact mode must be set (flag –105 clear). Numeric mode must not be set (flag –3 clear). The flags affect the output only if the input is not an integer.

Input: Level 1/Argument 1: An integer or real number.

Output: Level 1/Item 1: The integer converted to a real number.

See also: →NUM, R→I, XNUM

JORDAN

Type: Command

Description: Diagonalization, or Jordan cycle decomposition, of a matrix. Computes the eigenvalues, eigenvectors, minimum polynomial, and characteristic polynomial of a matrix.

Access: Matrices, ◁ MATRICES NXT EIGENVECTORS

Input: An $n \times n$ matrix.

Output: Level 4/Item 1: The minimum polynomial.
Level 3/Item 2: The characteristic polynomial.
Level 2/Item 3: A list of characteristic spaces tagged by the corresponding eigenvalue (either a

Example: Find the integral of sin(x) with respect to x, at the point where x=y.

Command: `INT(SIN(X),X,Y)`

Result: `-COS(Y)`

See also: INTVX, RISCH

INTEGER

Type: Command

Description: Displays a menu or list of CAS integer operations.

Access: Catalog, ▷ CAT

Flags: If the CHOOSE boxes flag is clear (flag –117 clear), displays the operations as a numbered list. If the flag is set, displays the operations as a menu of function keys.

See also: ALGB, ARIT, CONSTANTS, DIFF, EXP&LN, MAIN, MATHS, MATR, MODULAR, POLYNOMIAL, REWRITE, TESTS, TRIGO

INTVX

Type: Function

Description: Finds the antiderivative of a function symbolically, with respect to the current default variable.

Access: Calculus, ◁ CALC or SYMB CALC or ◁ CALC DERIV. & INTEG NXT

Input: An expression.

Output: The antiderivative of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Find the antiderivative of the following:
$x^2 \ln x$

Command: `INTVX(X^2*LN(X))`

Result: `1/3*X^3*LN(X)+(-1/9)*X^3`

See also: IBP, RISCH, PREVAL

INV

Type: Analytic function

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

Access: 1/X

Flags: Numerical Results (–3)

PERTBL

Type: Command

Description: Starts the Periodic Table. It doesn't affect the stack.

Access: APPS PERIODIC TABLE PERTBL

Flags: Units Usage (61), Units Type (60)

Input/Output: None

See also: MOLWT, PERINFO, PTPROP

PEVAL

Type: Command

Description: Polynomial Evaluation Command: Evaluates an *n*-degree polynomial at *x*.
The arguments must be an array of length *n* + 1 containing the polynomial's coefficients listed from highest order to lowest, and the value *x* at which the polynomial is to be evaluated.

Access: ↱ CAT PEVAL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
⌈ *array* ⌉coefficients	*x*	→	*p(x)*

Example: Find the polynomial $x^4 + 2x^3 - 25x^2 - 26x + 120$ at $x = 8$:

Command: `[ 1 2 -25 -26 120 ] 8 PEVAL`

Result: `3432`

See also: PCOEF, PROOT

PGDIR

Type: Command

Description: Purge Directory Command: Purges the named directory (whether empty or not).

Access: ↰ PRG MEMORY DIRECTORY PGDIR((PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*global*'	→	

See also: CLVAR, CRDIR, HOME, PATH, PURGE, UPDIR

PICK

Type: RPL Command

Description: Pick Object Command: Copies the contents of a specified stack level to level 1.

Access: ↰ PRG STACK PICK (PRG is the left-shift of the EVAL key).

Input/Output:

L_{n+1}...	L_2	L_1		L_{n+1}	L_2	L_1
obj_n ...	obj_1	*n*	→	obj_n ...	obj_1	obj_1

L = Level

See also: DUP, DUPN, DUP2, OVER, ROLL, ROLLD, ROT, SWAP

PICK3

Type: RPL Command

Description: Duplicates the object on level 3 of the stack.

Access: ↰ PRG STACK NXT PICK3 (PRG is the left-shift of the EVAL key).

Flags: Numerical Results (–3), Infinite Result Exception (–22)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*ln (x + 1)*
'*symb*'	→	'*LNP1(symb)*'

See also: EXPM, LN

LOCAL

Type: Command

Description: Creates one or more local variables. This command is intended mainly for use in Algebraic mode; it can not be single stepped when a program containing it is being debugged in Algebraic mode.

Access: Catalog, ↱ CAT

Input: Level 1/Argument 1: A list of one or more local variable names (names beginning with the local variable identifier ←), each one followed by an equals sign and the value to be stored in it. Any variable not followed by an equal sign and a value is set equal to zero.

Output: Level 1/Item 1: The input list.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Create local variables ←A and ←B and store the values 0 in the first and 2 in the second.

Command: `LOCAL({←A,←B=2})`

Result: `{←A,←B=2}`

See also: DEF, STORE, UNBIND

LOG

Type: Analytic function

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

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 the *general solution*:

```
LOG(Z)+2*π*i*n1/2.30258509299
```

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 calculator 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 $\log z = \ln z / \ln 10$.

Access: ↱ LOG (LOG is the right-shift of the EEX key).

Flags: Principal Solution (–1), Numerical Results (–3), Infinite Result Exception (–22)

PCONTOUR

Type: Command

Description: PCONTOUR Plot Type Command: Sets the plot type to PCONTOUR.

When plot type is set PCONTOUR, the DRAW command plots a contour-map view of a scalar function of two variables. PCONTOUR requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR is made up of the following elements:

{ x_{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 PCONTOUR, 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 not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

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

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

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

- (x_{min}, y_{min}) and (x_{max}, y_{max}) are not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command PCONTOUR places the name PCONTOUR in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: ▶ CAT PCONTOUR

Input/Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PCOV

Type: Command

Description: Population Covariance Command: Returns the population covariance of the independent and dependent data columns in the current statistics matrix (reserved variable *ΣDAT*).

The columns are specified by the first two elements in reserved variable *ΣPAR*, set by XCOL and YCOL respectively. If *ΣPAR* does not exist, PCOV creates it and sets the elements to their default values (1 and 2).

The population covariance is calculated with the following formula:

$$\frac{1}{n}\sum_{k=1}^{n}(x_{kn_1}-\overline{x_{n_1}})(x_{kn_2}-\overline{x_{n_2}})$$

where x_{kn_1} is the kth coordinate value in column n_1, x_{kn_2} is the kth coordinate value in the column n_2, $\overline{x_{n_1}}$ is the mean of the data in column n_1, $\overline{x_{n_2}}$ is the mean of the data in column n_2, and n is the number of data points.

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

Access: ▶ CAT LR

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
	→	*Intercept:* x_1	*Slope:* x_2

See also: BESTFIT, COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, PREDX, PREDY, PWRFIT, XCOL, YCOL

LSQ

Type: Command

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

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 $A \times X - B$.

Access: ◀ MATRICES OPERATIONS NXT LSQ (MATRICES is the left-shift of the 5 key).

◀ MTH MATRIX LSQ (MTH is the left-shift of the SYMB key).

Flags: Singular Values (–54)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[array]$_B$	*[[matrix]]*$_A$	→	*[array]*$_x$
[[matrix]]$_B$	*[[matrix]]*$_A$	→	*[[matrix]]*$_x$

See also: LQ, RANK, QR, /

LU

Type: Command

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

When solving an exactly determined system of equations, inverting a square matrix, or computing the determinant of a matrix, the calculator 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 $P \times A \equiv L \times U$.

Access: ◀ MATRICES FACTORIZATION LU (MATRICES is the left-shift of the 5 key).

◀ MTH MATRIX FACTOR LU (MTH is the left-shift of the SYMB key).

PARSURFACE

Type: Command

Description: PARSURFACE Plot Type Command: Sets plot type to PARSURFACE.

When plot type is set to PARSURFACE, the DRAW command plots an image graph of a 3-vector-valued function of two variables. PARSURFACE requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR is made up of the following elements:

{ x_{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 PARSURFACE, 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 that 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 the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

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

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

For plot type PARSURFACE, 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 not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command PARSURFACE places the name PARSURFACE in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: ⌐ CAT PARSURFACE

Input/Output: None

See also: BAR, CONIC, DIFFEQ, FAST3D, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PARTFRAC

Type: Command

Description: Performs partial fraction decomposition on a partial fraction.

Access: Algebra ⌐ ALG or Arithmetic, ¬ ARITH POLYNOMIAL NXT NXT

Input: An algebraic expression.

Output: The partial fraction decomposition of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Perform a partial fraction decomposition of the following expression:

$$\frac{1}{x^2-1}$$

Command: `PARTFRAC(1/(X^2-1))`

MAIN

Type: Command

Description: Displays the main menu (or list) of CAS operations. This displays the CASCFG command, the ALGB, ARIT, DIFF, EXP&LN, MATHS MATR, REWRITE and TRIGO menu commands described in this part of the Command Reference, and the CMPLX and SOLVER menu commands described in the Full Command and Function Reference (Chapter 3). Other menus are not shown because they are within the submenus given by MAIN. More details are given in Appendix K of the User's Guide.

Access: Catalog, ⌐ CAT

Flags: If the CHOOSE boxes flag is clear (flag –117 clear), displays the operations as a list. If the flag is set, displays the operations as a menu of function keys.

See also: ALGB, ARIT, CONSTANTS, DIFF, EXP&LN, INTEGER, MATHS, MATR, MODULAR, POLYNOMIAL, REWRITE, TESTS, TRIGO

MANT

Type: Function

Description: Mantissa Function: Returns the mantissa of the argument.

Access: ¬ MTH REAL NXT MANT (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3)

Input/Output:

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

See also: SIGN, XPON

MAP

Type: Command

Description: Applies a specified program to a list of objects or values. If one of the objects is a list, MAP will apply the program recursively to the items in the inner list.

- Level 1/Argument 2 contains the program to apply to the objects or values.
- Level 2/Argument 1 contains the list, matrix, or vector.

Access: ⌐ CAT MAP

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\{list\}_1$	«*program*»	→	$\{list\}_2$

Example: `{ 1 2 { 3 4 } } « →STR » MAP` returns `{ "1" "2" { "3" "4" } }`.

↓MATCH

Type: Command

Description: Match Pattern Down Command: Rewrites an expression that matches a specified pattern.

↓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.

Access: ← PRG STACK OVER (PRG is the left-shift of the EVAL key).

Input/Output:

Level 2	Level 1		Level 3	Level 2	Level 1
*obj*₁	*obj*₂	→	*obj*₁	*obj*₂	*obj*₁

See also: PICK, ROLL, ROLLD, ROT, SWAP

P2C

Type: Command

Description: Takes a list representing a permutation as an argument, and returns the permutation decomposed into lists that represent cycles.

Access: ← ARITH PERM

Input: A list representing a permutation. For example, the list {3,1,2,5,4} defines a permutation *P*, such that *P*(1)=3, *P*(2)=1, *P*(3)=2, *P*(4)=5, and *P*(5)=4

Output: Level 2/Item 1:A list of cycles equivalent to the permutation. For example, the list {3,1,2,5,4} defines a cycle *C*, such that *C*(3)=1, *C*(1)=2, *C*(2)=5, *C*(5)=4, and *C*(4)=3
Level 1, Item 2: The signature of the permutation, 1 or –1.

Example: Convert the permutation given by {3,4,5,2,1} into cycles:

Command: `P2C({3,4,5,2,1})`

Result: `{{{1,3,5},{2,4}},-1}`

See also: C2P, CIRC

PA2B2

Type: Function

Description: Takes a prime number, *p*, such that $p=2$ or $p \equiv 1$ modulo 4, and returns a Gaussian integer $a + ib$ such that $p = a^2 + b^2$. This function is useful for factorizing Gaussian integers.

Access: Arithmetic, ← ARITH INTEGER NXT

Input: A prime number, *p*, such that $p=2$ or $p \equiv 1$ modulo 4

Output: A Gaussian integer a+ib such that $p=a^2+b^2$

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Complex mode must be set (flag –103 set).

See also: GAUSS

PARAMETRIC

Type: Command

Description: Parametric Plot Type Command: Sets the plot type to PARAMETRIC.
When the plot type is PARAMETRIC, the DRAW command plots the current equation as a complex-valued function of one real variable. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

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

For plot type PARAMETRIC, 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) for the HP 48gII and (–6.5,–3.9) for the HP 50g and 49g+.

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 2/Item 1	Level 1/Item 2
'*symb*₁'	{ '*symb*ₚₐₜ', '*symb*ᵣₑₚₗ' }	→	'*symb*₂'	0/1
'*symb*₁'	{ '*symb*ₚₐₜ', '*symb*ᵣₑₚₗ', '*symb*ᶜₒₙᵈ' }	→	'*symb*₂'	0/1

Example 1: 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.

Example 2: This sequence: `'SIN(X+π)' ( 'SIN(&A+π)' '-SIN(&A)' ) ↑MATCH` returns `'-SIN(X)'` to level 2 and `1` to level 1.

Example 3: This sequence: `'W+√(SQ(5))' ( '√(SQ(&A))' '&A' '&A≥0' ) ↑MATCH` returns `'W+5'` to level 2 and `1` to level 1.

See also: ↓MATCH

MATHS

Type: Command

Description: Displays a menu or list of CAS mathematics submenus. Details are given in Appendix J of the User's Guide.

Access: Catalog, → CAT

Flags: If the CHOOSE boxes flag is clear (flag –117 clear), displays the submenus as a list. If the flag is set, displays the submenus as a menu of function keys.

See also: ALGB, ARIT, CONSTANTS, DIFF, EXP&LN, INTEGER, MAIN, MATR, MODULAR, POLYNOMIAL, REWRITE, TESTS, TRIGO

MATR

Type: Command

Description: Displays a menu or list containing the CAS commands for matrix operations.

Access: Catalog, → CAT

Flags: If the CHOOSE boxes flag is clear (flag –117 clear), displays the operations as a numbered list. If the flag is set, displays the operations as a menu of function keys.

See also: ALGB, ARIT, CONSTANTS, DIFF, EXP&LN, INTEGER, MAIN, MATHS, MODULAR, POLYNOMIAL, REWRITE, TESTS, TRIGO

MAX

Type: Function

Description: Maximum Function: Returns the greater of two inputs.

Access: ← MTH REAL MAX (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 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*)*

The character set in the HP 82240A Infrared Printer does not match the character set of the calculator:

- 24 characters in the calculator's character set are not available in the HP 82240A Infrared Printer. (From the table in Appendix J, these characters are numbers 129, 130, 143-157, 159, 166, 169, 172, 174, 184, and 185.) The HP 82240A prints a ▒ in substitution.
- Many characters in the extended character table (character codes 128 through 255) do not have the same character code. For example, the ※ character has code 171 in the calculator and code 146 in the HP 82240A Infrared Printer.

To use the CHR command to print extended characters with an HP 82240A Infrared Printer, first execute OLDPRT. The remapping string modified by OLDPRT is the second parameter in *PRTPAR*. This string (which is empty in the default state) changes the character code of each byte to match the codes in the HP 82240A Infrared Printer character table.

To cancel OLDPRT character mapping in order to print to an HP 82240B Infrared Printer, purge the *PRTPAR* variable. To print a string containing graphics data, disable OLDPRT.

Access: ▶ CAT OLDPRT

See also: CR, DELAY, PRLCD, PRST, PRSTC, PRVAR, PR1

OPENIO

Type: Command

Description: Open I/O Port Command: Opens a serial port using the I/O parameters in the reserved variable *IOPAR*.

Since all Kermit-protocol commands automatically effect an OPENIO first, OPENIO is not normally needed, but can be used if an I/O transmission does not work. OPENIO is necessary for interaction with devices that interpret a closed port as a break.

OPENIO is also necessary for the automatic reception of data into the input buffer using non-Kermit commands. If the port is closed, incoming characters are ignored. If the port is open, incoming characters are automatically placed in the input buffer (up to 255 characters). These characters can be detected with BUFLEN, and can be read out of the input buffer using SRECV.

If the port is already open, OPENIO does not affect the data in the input buffer. However, if the port is closed, executing OPENIO clears the data in the input buffer.

For more information, refer to the reserved variable *IOPAR* in appendix D, "Reserved Variables".

Access: ▶ CAT OPENIO

Flags: I/O Device (–33), I/O Device for Wire (–78)

Input/Output: None

See also: BUFLEN, CLOSEIO, SBRK, SRECV, STIME, XMIT

OR

Type: Function

Description: OR Function: Returns the logical OR of two arguments.

When the arguments are binary integers or strings, OR does a bit-by-bit (base 2) logical comparison.

- An argument that is a binary integer is treated as a sequence of bits as long as the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments as shown in the following table:

MEAN

Type: Command

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

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.

Access: ▶ CAT MEAN OR ▶ STAT Single-variable statistics, Mean

(STAT is the right-shift of the 5 key and always invokes a choose box).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{mean}
	→	$[x_{mean1}, x_{mean2}, \ldots, x_{meanm}]$

See also: BINS, MAXΣ, MINΣ, SDEV, TOT, VAR

MEM

Type: Command

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

The number returned is only a rough indicator of usable available memory, since recovery features (LASTARG= ◀ ANS , ▶ 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.

Access: ◀ PRG MEMORY MEM (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x

See also: BYTES

MENU

Type: Command Operation

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

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.

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* } or a name containing a list ($name_{definition}$). Either argument is stored in reserved variable *CST*, and the custom menu is subsequently displayed.

MENU takes *any* object as a valid argument and stores it in *CST*. However, the calculator can build a custom menu *only* if *CST* contains a list or a name containing a list. Thus, if an object other

The number of a character can be found by accessing the Characters tool (▶ CHARS) and highlighting that character. The number appears near the bottom of the screen. These are also listed in Appendix J of this manual.

Access: ◀ PRG TYPE NXT NUM (PRG is the left-shift of the EVAL key).
◀ PRG NXT CHARS NUM (PRG is the left-shift of the EVAL key).
▶ & EVAL NUM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"string"	→	*n*

See also: CHR, POS, REPL, SIZE, SUB

NUMX

Type: Command

Description: Number of X-Steps Command: Sets the number of *x*-steps for each *y*-step in 3D perspective plots.

The number of *x*-steps is the number of independent variable points plotted for each dependent variable point plotted. This number must be 2 or more. This value is stored in the reserved variable *VPAR*. YSLICE is the only 3D plot type that does not use this value.

Access: ▶ CAT NUMX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_x	→	

See also: NUMY

NUMY

Type: Command

Description: Number of Y-Steps Command: Sets the number of *y*-steps across the view volume in 3D perspective plots.

The number of *y*-steps is the number of dependent variable points plotted across the view volume. This number must be 2 or more. This value is stored in the reserved variable *VPAR*.

Access: ▶ CAT NUMY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_y	→	

See also: NUMX

OBJ→

Type: Command

Description: Object to Stack Command: Separates an object into its components. For some object types, the *number* of components is returned as item *n*+1 (stack level 1).

If the argument is a complex number, list, array, or string, OBJ→ provides the same functions as C→R, LIST→, ARRY→, and STR→, respectively. For lists, OBJ→ also returns the number of list elements. If the argument is an array, OBJ→ also returns the dimensions { *m n* } of the array, where *m* is the number of rows and *n* is the number of columns.

For algebraic objects, OBJ→ returns the arguments of the top-level (least-nested) function (arg_1 … arg_n), the number of arguments of the top-level function (*n*), and the name of the top-level function (*function*).

If the argument is a string, the object sequence defined by the string is executed.

Access: ◀ PRG TYPE OBJ→ (PRG is the left-shift of the EVAL key).

108-113	EXLR	LNAME	ADDTMOD	SUBTMOD	MULTMOD	DIVMOD
114-119	DIV2MOD	POWMOD	INVMOD	GCDMOD	EXPANDMOD	FACTORMOD
120-125	RREFMOD	MODSTO	MENUXY	KEYEVAL	GROBADD	SCROLL
126-131	CASCFG	MAIN	ALGB	CMPLX	TRIGO	MATR
132-137	DIFF	ARIT	SOLVER	EXP&LN	EPSX0	?
138-140	∞	PROMPTSTO	VER			

Example: Display a menu containing ATAN2S, ASIN2T, ASIN2C and ACOS2S.

Command: `MENUXY(34,37)`

Result: The four functions are displayed above the F1 to F4 keys. In Algebraic mode, NOVAL is returned as item 1.

See also: MENU, TMENU

MERGE

Type: Command

Description: Do not use this command, a carry-over from the HP 48SX for handling plug-in RAM cards.

Access: ▶ CAT MERGE

MIN

Type: Function

Description: Minimum Function: Returns the lesser of two inputs.

Access: ◀ MTH REAL MIN (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 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	→	$min(x\_unit_1, y\_unit_2)$

Example 1: `10 23 MIN` returns `10`.

Example 2: `-10 -23 MIN` returns `-23`.

Example 3: `1_m 9_cm MIN` returns `9_cm`.

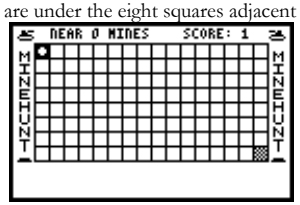
See also: MAX

MINEHUNT

Type: Command

Description: Starts the MINEHUNT game.

In this game, you are standing in the upper-left corner of an 8x16 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.



Command: `NEXTPRIME(145)`

Result: `149`

See also: ISPRIME?, PREVPRIME

NIP

Type: RPL command

Description: Drops the $(n–1)^{th}$ argument, where n is the number of arguments or items on the stack. (that is, the object on level 2 of the stack). This is equivalent to executing SWAP followed by DROP in RPN mode.

Access: ◁ PRG STACK NXT NXT NIP (PRG is the left-shift of the EVAL key).

TOOL STACK NXT NXT NIP

Input/Output:

Level 2	Level 1		Level 1
obj₁	*obj₂*	→	*obj₂*

Example: `333 222 1 NIP` returns `333 1`

See also: DUP, DUPDUP, DUPN, DUP2

NOT

Type: Function

Description: NOT Command: Returns the one's complement or logical inverse of the argument.

When the argument is a binary integer or string, NOT complements each bit in the argument to produce the result.

- A binary integer is treated as a sequence of bits as long as the current wordsize.
- A string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code).

When the argument is a real number or symbolic, NOT does a true/false test. The result is 1 (true) if the argument is zero; it is 0 (false) if the argument is nonzero. This test is usually done on a test result (T/F).

If the argument is an algebraic object, then the result is an algebraic of the form NOT *symb*. Execute →NUM (or set flag –3 before executing NOT) to produce a numeric result from the algebraic result.

Access: ◁ PRG TEST NXT NOT (PRG is the left-shift of the EVAL key).

▷ BASE NXT LOGIC NOT (BASE is the right-shift of the 3 key).

Flags: Numerical Results (–3), Binary Integer Wordsize (–5 through –10)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n_1	→	#n_2
T/F	→	*0/1*
"string₁"	→	*"string₂"*
'symb'	→	*'NOT symb'*

See also: AND, OR, XOR

NOVAL

Type: Command

Description: INFORM Place Holder/Result Command: Place holder for reset and initial values in user-defined dialog boxes. NOVAL is returned when a field is empty.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*'MINR'*
	→	*1.00000000000E–499*

See also: *e*, *i*, MAXR, π

MINΣ

Type: Command

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

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

Access: ▷ CAT MINΣ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{min}
	→	$\{ x_{min1}\ x_{min2} \ldots x_{min\,m} \}$

See also: BINS, MAXΣ, MEAN, SDEV, TOT, VAR

MITM

Type: Command

Description: Multiple-equation Menu Item Order Command. Changes multiple equation menu titles and order. The argument 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.

Access: ▷ CAT MITM

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"*title*"	{ *list* }	→	

See also: MINIT

MKISOM

Type: Command

Description: Returns the matrix representation for a given isometry.

Access: Matrices, ◁ MATRICES LINEAR APPL

Input: Level 2/Argument 1: For a 3-d isometry, a list of the characteristic elements of the isometry. For a 2-d isometry, the characteristic element of the isometry (either an angle or a vector).
Level 1/Argument 2: +1 for a direct isometry or –1 for an indirect isometry.

Output: The matrix that represents the given isometry.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example 1: Find the matrix for a rotation of π/2 radians in two dimensions

Command: `MKISOM(π/2, 1)`

Result:

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

NDIST

Type: Command

Description: Normal Distribution Command: Returns the normal probability distribution (bell curve) at x based on the mean m and variance v of the normal distribution.
NDIST is calculated using this formula:

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

Access: ◀ MTH ▶ PROBABILITY ▶ NDIST (MTH is the left-shift of the SYMB key).

Input/Output:

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

See also: UTPN

NDUPN

Type: RPL command

Description: Duplicates an object *n* times, and returns *n*.

Access: ◀ PRG STACK ◀ PREV NDUPN (PRG is the left-shift of the EVAL key).
TOOL STACK ◀ PREV NDUPN

Input/Output:

Level 2	Level 1		Level$_{n+1}$ … Level$_2$	Level$_1$
obj	*n*	→	*obj … obj*	*n*

Example: To make a list of 100 "X"s, run `"X" 100 NDUPN →LIST`.

See also: DUP, DUPDUP, DUPN, DUP2

NEG

Type: Analytic function

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

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.

Access: ▶ CMPLX NEG (CMPLX is the right-shift of the I key).
◀ MTH ▶ COMPLEX ▶ NEG (MTH is the left-shift of the SYMB key).
+/–

Flags: Numerical Results (–3), Binary Integer Wordsize (–5 through –10)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*–z*
#n$_1$	→	*#n*$_2$
[array]	→	*[–array]*
'symb'	→	*'–(symb)'*
x_unit	→	*–x_unit*
grob$_1$	→	*grob*$_2$
PICT$_1$	→	*PICT*$_2$

MOLWT

Type: Function

Description: Returns the molecular weight for the specified molecular formula. It takes the formula as a string (such as "H2O") or name (with certain restrictions, such as 'H2O'). It returns the molecular weight. It chooses to use or not use units according to the Units Usage flag (flag 61: SI units if clear, no units if set).

You can store a molecular formula in a variable, then use the variable name with MOLWT. You should do this when you want to use MOLWT in an expression and the formula contains parentheses or matches a command name. You must take care when naming a variable that contains a formula string or name. Make sure the variable name isn't a valid formula — for example, start the variable name with a lowercase letter. (If the variable name is a valid formula, using MOLWT with the variable name returns the molecular weight for the variable name, not for the formula it contains.)

Access: APPS PERIODIC TABLE MOLWT

Flags: Units Usage (61)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'name'	→	*x* or *x_unit*
"string"	→	*x* or *x_unit*

Example 1: The command sequence `"CH3C6H2(NO2)3" MOLWT` returns `'227.133_g/gmol'` when flag 61 is clear.

Example 2: The command sequence `'C12H17C1N4OS' MOLWT` returns `300.8055` when flag 61 is set.

See also: PERINFO, PERTBL, PTPROP

MROOT

Type: Command

Description: Multiple Roots Command: Uses the multiple-equation solver to solve for one or more variables using the equations in *EQ*. Given a variable name, MROOT returns the found value; with "ALL" MROOT stores a found value for each variable but returns nothing.

Access: ▶ CAT MROOT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'name'	→	*x*
"ALL"	→	

See also: MCALC, MUSER

MSGBOX

Type: Command

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

MSGBOX displays "*message*" in the form of a standard message box. Message text too long to appear on the screen is truncated. You can use spaces and new-line characters (▶ ENTER) to control word-wrapping and line breaks within the message.

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

Access: ◀ PRG ▶ OUT MSGBOX (PRG is the left-shift of the EVAL key).

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[[matrix]]₁	$n_{position}$	*[[matrix]]₂*	→	*[[matrix]]₃*
[[matrix]]₁	{ n_{row}, n_{column} }	*[[matrix]]₂*	→	*[[matrix]]₃*
[vector]₁	$n_{position}$	*[vector]₂*	→	*[vector]₃*
{ $list_{target}$ }	$n_{position}$	{ $list_1$ }	→	{ $list_{result}$ }
"$string_{target}$"	$n_{position}$	"$string_1$"	→	"$string_{result}$"
$grob_{target}$	*(#n, #m)*	$grob_1$	→	$grob_{result}$
$grob_{target}$	*(x,y)*	$grob_1$	→	$grob_{result}$
PICT	*(#n, #m)*	$grob_1$	→	
PICT	*(x,y)*	$grob_1$	→	

Example 1: `[[ 1 1 1 1 ][ 1 1 1 1 ][ 1 1 1 1 ]] 6 [[ 2 2 ][ 2 2 ]] REPL`
returns `[[ 1 1 1 1 ][ 1 2 2 1 ][ 1 2 2 1 ]]`.

Example 2: `« A B C D E » 2 « F G » REPL` returns `« A F G D E »`.

Example 3: `ERASE PICT (0,0) # 5d # 5d BLANK NEG REPL` replaces a portion of *PICT* with a 5 x 5 graphics object, each of whose pixels is on (dark), and whose upper left corner is positioned at (0,0) in *PICT*.

See also: CHR, GOR, GXOR, NUM, POS, SIZE, SUB

RES

Type: Command

Description: Resolution Command: Specifies the resolution of mathematical and statistical plots, where the resolution is the interval between values of the independent variable used to generate the plot. A real number $n_{interval}$ specifies the interval in user units. A binary integer $\#n_{interval}$ specifies the interval in pixels.
The resolution is stored as the fourth item in *PPAR*, with default value 0. The interpretation of the default value is summarized in the following table.

Plot Type	Default Interval
BAR	10 pixels (bar width = 10 pixel columns)
DIFFEQ	unlimited: step size is not constrained
FUNCTION	2 pixels (plots a point in every other column of pixels)
CONIC	2 pixels (plots a point in every other column of pixels)
TRUTH	2 pixels (plots a point in every other column of pixels)
GRIDMAP	RES does not apply
HISTOGRAM	10 pixels (bin width = 10 pixel columns)
PARAMETRIC	[independent variable range in user units]/130
PARSURFACE	RES does not apply

Input/Output:

L_3	L_2	L_1		L_4	L_3	L_2	L_1
obj_1	obj_2	obj_3	→	obj_1	obj_2	obj_3	obj_1

L = Level; A = Argument; I =Item

Example: `333 22 1 PICK3` returns `333 22 1 333`.

See also: PICK, OVER, DUP

PICT

Type: Command

Description: PICT Command: Puts the name PICT on the stack.

PICT is the name of a storage location in calculator memory containing the current graphics object. The command PICT enables access to the contents of that memory location as if it were a variable. Note, however, that *PICT* is *not* a variable as defined in the calculator: its name cannot be quoted, and only graphics objects may be stored in it.

If a graphics object smaller than 131 wide × 80 pixels high is stored in *PICT*, it is enlarged to 131 × 80. (These values are 131 x 64 on the HP 48gII). A graphics object of unlimited pixel height and up to 2048 pixels wide can be stored in *PICT*.

Access: ◀ PRG ▶ NXT [PICT] PICT (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*PICT*

Example: `PICT RCL` returns the current graphics object to the stack.

See also: GOR, GXOR, NEG, PICTURE, PVIEW, RCL, REPL, SIZE, STO, SUB

PICTURE

Type: Command

Description: Picture Environment Command: Selects the Picture environment (that is, selects the graphics display and activates the graphics cursor and Picture menu).
When executed from a program, PICTURE suspends program execution until CANCEL is pressed.

Access: ▶ CAT PICTURE
◀

Input/Output: None

Example: This program:

```
« "Press CANCEL to return↵to stack"
  1 DISP 3 WAIT PICTURE »
```

displays a message for 3 seconds, then selects the Picture environment. (The ↵ character in the program indicates a linefeed.)

See also: PICT, PVIEW, TEXT

PINIT

Type: Command

Description: Port Initialize Command: Initializes all currently active ports. It may affect data already stored in a port.
PINIT is particularly useful when a third-party library has corrupted memory. It stores and then purges an object in each internal port. This has the effect of initializing each port without disturbing any previous-stored data, while removing any invalid objects.

Access: ▶ CAT PINIT

Flags: I/O Device flag (–33), I/O Data Format (–35), RECV Overwrite (–36), I/O Messages (–39), I/O Device for Wire (–78)

Input/Output: None

See also: BAUD, CKSM, FINISH, KGET, PARITY, RECN, SEND, SERVER, TRANSIO

REF

Type: Command

Description: Reduces a matrix to echelon form. This is a subdiagonal reduction (Gauss, not Gauss-Jordan).

Access: Matrices, ◀ MATRICES LINEAR SYSTEMS

Input: A real or complex matrix.

Output: The equivalent matrix in echelon form.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Complex mode must be set (flag –103 set) if the input is complex.

See also: rref, RREFMOD

REMAINDER

Type: Function

Description: Returns the remainder of the Euclidean division of two polynomials.

Access: Arithmetic, ◀ ARITH POLYNOMIAL ◀ PREV

Input: Level 2/Argument 1: The numerator polynomial.
Level 1/Argument 2: The denominator polynomial.

Output: The remainder resulting from the Euclidean division.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
Complex mode must be set (flag –103 set) if either input is complex.

Example: Find the remainder of the division of $x^3 + 6x^2 + 11x + 6$ by $x^2 + 5x + 6$.

Command: `REMAINDER(X^3+6*X^2+11*X+6, X^2+5*X+6)`

Result: `0`

See also: QUOT

RENAME

Type: Command

Description: Rename Object Command: Renames an object to the name that you specify.

Access: ▶ CAT RENAME

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
new 'name'	*old 'name'*	→	

See also: COPY

Flags: I/O Device (–33), I/O Messages (–39), I/O Device for Wire (–78). The I/O Data Format flag (–35) can be significant if the server sends back more than one packet.

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"data"	*"type"*	→	*"response"*

Example 1: A PKT command to send a *g*eneric *d*irectory request is `"D" "G" PKT`.

Example 2: To send a *host command* packet, use a command from the server's operating system for the *data* string and `"C"` for the *type* string. For example, `"'ABC' PURGE" "C" PKT` on a local calculator would instructor a server calculator to purge variable *ABC*.

See also: CLOSEIO, KERRM, SERVER

PLOT

Type: Command

Description: Stores its argument in *EQ* and opens the PLOT SETUP screen.

Access: SYMB GRAPH PLOT

Input: An expression.

Output: The input expression.

Example: Store SIN(*X*) in EQ and open the PLOT SETUP screen:

Command: `PLOT(SIN(X))`

Result: PLOT SETUP screen is open with SIN(*X*) in *EQ*. SIN(*X*) is copied to history (LASTARG in RPN mode).

See also: PLOTADD

PLOTADD

Type: Function

Description: Adds a function to the existing plot function list, and opens the Plot Setup screen.

Access: SYMB GRAPH PLOTA

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(symb)	→	

PMAX

Type: Command

Description: PICT Maximum Command: Specifies (*x*, *y*) as the coordinates at the upper right corner of the display.
The complex number (*x*, *y*) is stored as the second element in the reserved variable *PPAR*.

Access: ▶ CAT PMAX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x,y)	→	

See also: PDIM, PMIN, XRNG, YRNG

PMIN

Type: Command

Description: PICT Minimum Command: Specifies (*x*, *y*) as the coordinates at the lower left corner of the display. The complex number (*x*, *y*) is stored as the first element in the reserved variable *PPAR*.

If the list contains a single number $n_{elements}$, the result is an n-element vector. If the list contains two numbers n_{rows} and m_{cols}, the result is an $n \times m$ matrix.

Elements taken from the argument vector or matrix preserve the same row order in the resulting vector or matrix. If the result is dimensioned to contain fewer elements than the argument vector or matrix, excess elements from the argument vector or matrix at the end of the row order are discarded. If the result is dimensioned to contain more elements than the argument vector or matrix, the additional elements in the result at the end of the row order are filled with zeros.

If the argument vector or matrix is specified by *global*, the result replaces the argument as the contents of the variable.

Access: ◀ MATRICES CREATE NXT NXT RDM (MATRICES is the left-shift of the 5 key).
◀ MTH MATRIX MAKE RDM (MTH is the left-shift of the SYMB key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
⌈ vector ⌋$_1$	{ $n_{elements}$ }	→	⌈ vector ⌋$_2$
⌈ vector ⌋	{ n_{rows}, m_{cols} }	→	⟦⌈ matrix ⌋⟧
⟦⌈ matrix ⌋⟧	{ $n_{elements}$ }	→	⌈ vector ⌋
⟦⌈ matrix ⌋⟧$_1$	{ n_{rows}, m_{cols} }	→	⟦⌈ matrix ⌋⟧$_2$
'global'	{ $n_{elements}$ }	→	
'global'	{ n_{rows}, m_{cols} }	→	

Example 1: `[ 2 4 6 8 ] < 2 2 > RDM` returns `[[ 2 4 ][ 6 8 ]]`.
Example 2: `[[ 2 3 4][ 1 6 9 ]] 8 RDM` returns `[ 2 3 4 1 6 9 0 0 ]`.
See also: TRN

RDZ

Type: Command
Description: Randomize Command: Uses a real number x_{seed} as a seed for the RAND command.

If the argument is 0, a random value based on the system clock is used as the seed.

Access: ◀ MTH NXT PROBABILITY RDZ (MTH is the left-shift of the SYMB key).
Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{seed}	→	

See also: COMB, PERM, RAND, !

RE

Type: Function
Description: Real Part Function: Returns the real part of the argument.

If the argument is a vector or matrix, RE returns a real array, the elements of which are equal to the real parts of the corresponding elements of the argument array.

Access: ◀ CMPLX NXT RE (CMPLX is the right-shift of the 1 key).
Flags: Numerical Results (–3)

- *res* is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is 0, which specifies an interval of 2 degrees, 2 grads, or π/90 radians.
- *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 POLAR places the name POLAR in *ptype*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The current equation is plotted as a function of the variable specified in *indep*. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the default minimum value is 0 and the default maximum value corresponds to one full circle in the current angle mode (360 degrees, 400 grads, or 2π radians). Lines are drawn between plotted points unless flag –31 is set.

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

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

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

Access: ◀ CAT POLAR
Flags: Simultaneous Plotting (–28), Curve Filling (–31)
Input/Output: None
See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

POLYNOMIAL

Type: Command
Description: Displays a menu or list of CAS operations with polynomials.
Access: Catalog, ◀ CAT
Flags: If the CHOOSE boxes flag is clear (flag –117 clear), displays the operations as a numbered list. If the flag is set, displays the operations as a menu of function keys.
See also: ALGB, ARIT, CONSTANTS, DIFF, EXP&LN, INTEGER, MAIN, MATHS, MATR, MODULAR, REWRITE, TESTS, TRIGO

POP

Type: Command
Description: Restores the flags and current directory saved by the most recent execution of PUSH. If no PUSH saves are left, the command has no effect.
Access: ◀ CAT POP
Input: None
Output: In Algebraic mode the command returns NOVAL to level 1 of the stack.
See also: PUSH, RCLF, STOF

Access: ↱ TIME Tools ALRM RCLALARM (TIME is the right-shift of the 9 key).
↱ & 9 ALRM RCLALARM
↰ PRG NXT NXT TIME ALRM RCLALARM (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{index}	→	{ date time obj_{action} x_{repeat} }

See also: DELALARM, FINDALARM, STOALARM

RCLF

Type: Command

Description: Recall Flags Command: Returns a list of integers representing the states of the system and user flags, respectively.

A bit with value 1 indicates that the corresponding flag is set; a bit with value 0 indicates that the corresponding flag is clear. The rightmost (least significant) bit of #n_{system} and #n_{user} indicate the states of system flag –1 and user flag +1, respectively.

Used with STOF, RCLF lets a program that alters the state of a flag or flags during program execution preserve the pre-program-execution flag status.

Access: ↰ & MODE FLAG NXT RCLF
↰ PRG NXT MODES FLAG NXT RCLF (PRG is the left-shift of the EVAL key).

Flags: Binary Integer Wordsize (–5 through –10)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	{ #n_{system} #n_{user} #$n_{system2}$ #n_{user2} }

See also: STOF, PUSH, POP

RCLKEYS

Type: Command

Description: Recall Key Assignments Command: Returns the current user key assignments. This includes an S if the standard definitions are active (not suppressed) for those keys without user key assignments.

The argument x_{key} is a real number of the form *r*:*p* specifying the key by its row number *r*, its column number *c*, and its plane (shift) *p*. (For a definition of plane, see the entry for ASN.)

Access: ↰ & MODE KEYS RCLKEYS
↰ PRG NXT MODES KEYS RCLKEYS (PRG is the left-shift of the EVAL key).

Flags: User-Mode Lock (–61) and User Mode (–62) affect the status of the user keyboard

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	{ obj_1, $x_{key\ 1}$, ... ,obj_n, $x_{key\ n}$ }
	→	{ *S*, obj_1, $x_{key\ 1}$, ... obj_n, $x_{key\ n}$ }

See also: ASN, DELKEYS, STOKEYS

RCLMENU

Type: Command

Description: Recall Menu Number Command: Returns the menu number of the currently displayed menu.

x_{menu} has the form *mm.pp*, where *mm* is the menu number and *pp* is the page of the menu.

Output: The result from applying the distributive property of exponentiation over multiplication.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Expand $(X+1)^3$.

Command: `POWEXPAND((X+1)^3)`

Result: `(X+1)·(X+1)·(X+1)`

POWMOD

Type: Function

Description: Raises an object (number or expression) to the specified power, and expresses the result modulo the current modulus.

Access: Arithmetic, ↰ ARITH MODULO NXT

Input: Level 2/Argument 1: The object.
Level 1/Argument 2: The exponent.

Output: The result of the object raised to the exponent, modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

PR1

Type: Command

Description: Print Level 1 Command: Prints an object in multiline printer format.

All objects except strings are printed with their identifying delimiters. Strings are printed without the leading and trailing " delimiters.

If flag –34 is set (printer output directed to the serial port), flag –33 must be clear.

Multiline printer format is similar to multiline display format, with the following exceptions:

- Strings and names that are more than 24 characters long are continued on the next printer line.
- The real and imaginary parts of complex numbers are printed on separate lines if they don't fit on the same line.
- Grobs are printed graphically.
- Arrays are printed with a numbered heading for each row and with a column number before each element.

For example, the 2 × 3 array

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

would be printed as follows:

```
Array ⟨ 2 3 ⟩
Row 1
1] 1
2] 2
3] 3

Row 2
1] 4
2] 5
3] 6
```

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
z_1	z_2	→	z_1/z_2
⌈ array ⌋	{⌈ matrix ⌋⌉	→	⟦⌈ array × matrix^{-1}⌋⟧
⌈ array ⌋	z	→	⌈ array/z ⌋
z	'symb'	→	'z/symb'
'symb'	z	→	'symb/z'
'symb$_1$'	'symb$_2$'	→	'symb$_1$/symb$_2$'
#n$_1$	n$_2$	→	#n$_3$
n$_1$	#n$_2$	→	#n$_3$
#n$_1$	#n$_2$	→	#n$_3$
x_unit$_1$	y_unit$_2$	→	(x/y)_unit$_1$/unit$_2$
x	y_unit	→	(x/y)_1/unit
x_unit	y	→	(x/y)_unit
'symb'	x_unit	→	'symb/x_unit'
x_unit	'symb'	→	'x_unit/symb'

See also: /

RCEQ

Type: Command

Description: Recall from EQ Command: Returns the unevaluated contents of the reserved variable *EQ* from the current directory.
To recall the contents of *EQ* from a parent directory (when *EQ* doesn't exist in the current directory) evaluate the name *EQ*.

Access: ↱ CAT RCEQ (or ↱ EQ after pressing VAR)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	obj$_{EQ}$

See also: STEQ

RCI

Type: Command

Description: Multiply Row by Constant Command: Multiplies row *n* of a matrix (or element *n* of a vector) by a constant x_{factor}, and returns the modified matrix.
RCI rounds the row number to the nearest integer, and treats vector arguments as column vectors.

Access: ↰ MATRICES CREATE ROW RCI (MATRICES is the left-shift of the 5 key).
↰ MTH MATRIX ROW RCI (MTH is the left-shift of the SYMB key).

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
⟦⌈ matrix ⌋⟧$_1$	x$_{factor}$	n$_{row\ number}$	→	⟦⌈ matrix ⌋⟧$_3$
⌈ vector ⌋$_1$	x$_{factor}$	n$_{element\ number}$	→	⌈ vector ⌋$_2$

See also: RCIJ

RCIJ

Type: Command

Description: Add Multiplied Row Command: Multiplies row *i* of a matrix by a constant x_{factor}, adds this product to row *j* of the matrix, and returns the modified matrix; or multiplies element *i* of a vector

PREDY

Type: Command

Description: Predicted y-Value Command: Returns the predicted dependent-variable value $y_{dependent}$, based on the independent-variable value $x_{independent}$, the currently selected statistical model, and the current regression coefficients in the reserved variable *ΣPAR*.

The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable *ΣPAR*. For the linear statistical model, the equation used is this:

$$y_{dependent} = (m x_{independent}) + b$$

where *m* is the slope (the third element in *ΣPAR*) and *b* is the intercept (the fourth element in *ΣPAR*).

For the other statistical models, the equations used by PREDY are listed in the LR entry.

If PREDY is executed without having previously generated regression coefficients in *ΣPAR*, a default value of zero is used for both regression coefficients–in this case PREDY will return 0 for statistical models LINFIT and LOGFIT, and error for statistical models EXPFIT and PWRFIT.

Access: ↱ CAT PREDY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$x_{independent}$	→	$y_{dependent}$

Example: Given four columns of data in *ΣDAT*, the command sequence:

```
2 XCOL 4 YCOL PWRFIT LR 11 PREDY
```

sets column 2 as the independent variable column, sets column 4 as the dependent variable column, and sets the power statistical model. It then executes LR, generating intercept and slope regression coefficients, and storing them in *ΣPAR*. Then, given an independent value of 11, it returns a predicted dependent value based on the regression coefficients and the statistical model.

See also: COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, LR, PREDX, PWRFIT, XCOL, YCOL

PREVAL

Type: Function

Description: With respect to the current default variable, returns the difference between the values of a function at two specified values of the variable.

PREVAL can be used in conjunction with INTVX to evaluate definite integrals. See the example below.

Access: Calculus, ↰ CALC DERIV. & INTEG NXT.

Input: Level 3/Argument 1: A function.
Level 2/Argument 2: The lower bound.
Level 3/Argument 1: The upper bound.
The bounds can be expressions.

Output: The result of the evaluation.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Evaluate the following:

$$\int_0^3 (x^3 + 3x)\,dx$$

Command:

```
PREVAL(INTVX(X^3+3*X),0,3)
```

QXA

Type: Command

Description: Expresses a quadratic form in matrix form.

Access: ◀┐ MATRICES QUADF, ◀┐ CONVERT MATRX

Input: Level 2/Argument 1: A quadratic form.
Level 1/Argument 2: A vector containing the variables.

Output: Level 2/Item 1: The quadratic form expressed in matrix form.
Level 1/Item 2: The vector containing the variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Express the following quadratic form in matrix form:
$x^2 + xy + y^2$

Command: `QXA(X^2+X*Y+Y^2, [X,Y])`

Result: `{[[1,1/2][1/2,1]],[X,Y]}`

See also: AXQ, GAUSS, SYLVESTER

RAD

Type: Command

Description: Radians Mode Command: Sets Radians angle mode.

RAD sets flag –17 and clears flag –18, and displays the RAD annunciator.

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

Access: ◀┐ & MODE ANGLE RAD
◀┐ PRG NXT MODES ANGLE RAD (PRG is the left-shift of the EVAL key).

Input/Output: None

See also: DEG, GRAD

RAND

Type: Command

Description: Random Number Command: Returns a pseudo-random number generated using a seed value, and updates the seed value.

The calculator uses a linear congruential method and a seed value to generate a random number x_{random} in the range $0 \le x < 1$. Each succeeding execution of RAND returns a value computed from a seed value based upon the previous RAND value. (Use RDZ to change the seed.)

Access: ◀┐ MTH NXT PROBABILITY RAND (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{random}

See also: COMB, PERM RDZ, !

RANK

Type: Command

Description: Matrix Rank Command: Returns the rank of a rectangular matrix.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"global"	→	

See also: PROMPT, STO

PROOT

Type: Command

Description: Polynomial Roots Command: Returns all roots of an n-degree polynomial having real or complex coefficients.

For an n^{th}-order polynomial, the argument must be a real or complex array of length $n+1$ containing the coefficients listed from highest order to lowest. The result is a real or complex vector of length n containing the computed roots.

PROOT interprets leading coefficients of zero in a limiting sense. As a leading coefficient approaches zero, a root of the polynomial approaches infinity: therefore, if flag –22 is clear (the default), PROOT reports an Infinite Result error if a leading coefficient is zero. If flag –22 is set, PROOT returns a root of (MAXREAL,0) for each leading zero in an array containing real coefficients, and a root of (MAXREAL,MAXREAL) for each leading zero in an array containing complex coefficients.

Access: ◀┐ ARITH POLYNOMIAL NXT NXT PROOT (ARITH is the left-shift of the | key).

Flags: Infinite Result Exception (–22)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[*array*]$_{coefficients}$	→	[*array*]$_{roots}$

Example: Find the roots of the polynomial $x^4 + 2x^3 - 25x^2 - 26x + 120$:

Command: `[ 1 2 -25 -26 120 ] PROOT`

Result: `[ 2 -3 4 -5 ]`

See also: PCOEF, PEVAL

PROPFRAC

Type: Command

Description: Toggles between an improper fraction and its corresponding integer and fractional part.

Access: SYMB ARITH or Arithmetic, ◀┐ ARITH NXT

Input: An improper fraction, or an object that evaluates to an improper fraction. It must not contain real numbers. Alternately, the input may be an integer part plus a proper fraction.

Output: An integer part plus a proper fraction; or alternately, if the input was an integer part plus a proper fraction, an improper fraction.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Express the following as a proper fraction:

$$\frac{x^3+4}{x^2}$$

Command: `PROPFRAC((X^3+4)/X^2))`

Result: `X+(4/X^2)`

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	'a/b*π'
x	→	'a/b'
'symb$_1$'	→	'symb$_2$'
(x,y)	→	'a/b*π + c/d*π*i'
(x,y)	→	'a/b + c/d*i'

Example: In Fix mode to three decimal places, `6.2832 →Q π` returns `'44/7'`. In Standard mode, however, `6.2832 →Q π` returns `'3927/625'`.

See also: →Q, /, XQ, π

qr

Type: Command

Description: qr Factorization of a square Matrix Command: Returns the qr factorization of an $n \times n$ matrix.
qr factors an $n \times n$ matrix A into two matrices:
- Q is an $n \times m$ orthogonal matrix.
- R is an $n \times n$ triangular matrix.

Where $A = Q \times R$.

Access: ◄ MATRICES FACTORIZATION qr (MATRICES is the left-shift of the 5 key).

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
[[matrix]]$_A$	→	[[matrix]]$_Q$	[[matrix]]$_R$

See also: LQ, LSQ

QR

Type: Command

Description: QR Factorization of a Matrix Command: Returns the QR factorization of an $m \times n$ matrix.
QR factors an $m \times n$ matrix A into three matrices:
- Q is an $m \times m$ orthogonal matrix.
- R is an $m \times n$ upper trapezoidal matrix.
- P is a $n \times n$ permutation matrix.

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

Access: ◄ MATRICES FACTORIZATION QR (MATRICES is the left-shift of the 5 key).
◄ MTH MATRIX FACTORS QR (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 3/Item 1	Level 2/Item 2	Level 1/Item 3
[[matrix]]$_A$	→	[[matrix]]$_Q$	[[matrix]]$_R$	[[matrix]]$_P$

See also: LQ, LSQ

QUAD

Type: Command

Description: Solve Quadratic Equation Command: This command is identical to the computer algebra command SOLVE, and is included for backward compatibility with the HP 48G series.

Access: ◄ CAT QUAD

Flags: Principal Solution (–1)

PSDEV

Type: Command

Description: Population Standard Deviation Command: Calculates the population standard deviation of each of the m columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).
PSDEV returns a vector of m real numbers, or a single real number if $m = 1$. The population standard deviation is computed using this formula:

$$\sqrt{\frac{1}{n}\sum_{k=1}^{n}(x_k - \bar{x})^2}$$

where x_k is the kth coordinate value in a column, $\bar{x}$ is the mean of the data in this column, and n is the number of data points.

Access: ◄ CAT PSDEV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{psdev}
	→	$\{ x_{psdev1}\ x_{psdev2} \ldots x_{psdevm} \}$

See also: MEAN, PCOV, PVAR, SDEV, TOT, VAR

PSI

Type: Function

Description: Calculates the polygamma function, the nth derivative of the digamma function, at a point a.
PSI(a, 0) is equivalent to Psi(a).

Access: ◄ MTH NXT SPECIAL

Input: Level 2/Argument 1: A real or complex expression specifying the point a.
Level 1/Argument 2: A non-negative integer, n.

Output: The value of the polygamma function PSI(a, n).

Flags: Exact mode must be set (flag –105 clear), and
numeric mode must not be set (flag –3 clear), if symbolic results are wanted.
Complex mode must be set (flag –103 set) if a complex value is used for point a.

See also: Psi

Psi

Type: Function

Description: Calculates the digamma function at a point a. The digamma function is the derivative of the natural logarithm (ln) of the gamma function. The function can be represented as follows:

$$\Psi(z) = \frac{d}{dz}(\ln \Gamma(z)) = \frac{\Gamma'(z)}{\Gamma(z)}$$

Access: ◄ MTH NXT SPECIAL

Input: A real or complex expression specifying the point a.

Output: The digamma function at the specified point.

Flags: Exact mode must be set (flag –105 clear), and
numeric mode must not be set (flag –3 clear), if symbolic results are wanted. For example, with these settings, Psi(3) evaluates to the symbolic value Psi(3).
Complex mode must be set (flag –103 set) if a complex value is used for point a.

See also: PSI

If n_{port} = 0, then *memory* is bytes of available main RAM; otherwise *memory* is bytes of available RAM in the specified port.

Access: [→] [CAT] PVARS

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
n_{port}	→	{ $:n_{port}$ $:name_{backup}$ ... }	*memory*
n_{port}	→	{ $:n_{port}$ $:n_{library}$ ... }	*memory*

See also: PVARS, VARS

PVIEW

Type: Command

Description: PICT View Command: Displays *PICT* with the specified coordinate at the upper left corner of the graphics display.

PICT must fill the entire display on execution of PVIEW. Thus, if a position other than the upper left corner of *PICT* is specified, *PICT* must be large enough to fill a rectangle that extends 131 pixels to the right and 80 pixels down on the HP 50g and 49g+ (64 pixels down on the HP 48gII).

If PVIEW is executed from a program with a coordinate argument (versus an empty list), the graphics display persists only until the keyboard is ready for input (for example, until the end of program execution). However, the FREEZE command freezes the display until a key is pressed.

If PVIEW is executed with an *empty* list argument, *PICT* is centered in the graphics display with scrolling mode activated. In this case, the graphics display persists until [CANCEL] is pressed.

PVIEW does *not* activate the graphics cursor or the Picture menu. To activate the graphics cursor and Picture menu, execute PICTURE.

Access: [→] [PRG] [NXT] PICT [NXT] PVIEW ([PRG] is the left-shift of the [EVAL] key).

[→] [PRG] [NXT] OUT PVIEW ([PRG] is the left-shift of the [EVAL] key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x,y)	→	
{ #*n*. #*m* }	→	
{ }	→	

Example: The program

```
« { # 0d # 0d } PVIEW 7 FREEZE »
```

displays *PICT* in the graphics display with coordinates { # 0d # 0d } in the upper left corner of the display, then freezes the full display until a key is pressed.

See also: FREEZE, PICTURE, TEXT

PWRFIT

Type: Command

Description: Power Curve Fit Command: Stores PWRFIT as the fifth parameter in the reserved variable *ΣPAR*, indicating that subsequent executions of LR are to use the power curve fitting model. LINFIT is the default specification in *ΣPAR*.

Access: [→] [CAT] PWRFIT

Input/Output: None

See also: BESTFIT, EXPFIT, LINFIT, LOGFIT, LR

number can be replaced with the wildcard character &, in which case the calculator will search ports 0 through 2, and then main memory for the named backup object.

A library object must be detached before it can be purged from the *HOME* directory.

Neither a library object nor a backup object can be purged if it is currently "referenced" internally by stack pointers (such as an object on the stack, in a local variable, on the LAST stack, or on an internal return stack). This produces the error Object in Use. To avoid these restrictions, use NEWOB before purging. (See NEWOB.)

Access: [→] [PRG] MEMORY PURGE ([PRG] is the left-shift of the [EVAL] key).

[TOOL] PURGE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*global*'	→	
{ $global_1$ ... $global_n$ }	→	
PICT	→	
$:n_{port}$ $:name_{backup}$	→	
$:n_{port}$ $:n_{library}$	→	

See also: CLEAR, CLVAR, NEWOB, PGDIR

PUSH

Type: Command

Description: Saves the current status of the flags, and the current directory path. This allows the user to change the flags or the directory path, then restore them all with the command POP. PUSH is equivalent to saving the results of the commands RCLF and PATH, but it saves them in a stack from which the most recently saved values are recovered by POP, with no need to use named variables. The flags and the path are stored in the CASDIR directory, as a list of lists, in the variable ENVSTACK.

Access: [→] [CAT] PUSH

Input: None.

Output: Item 1: In Algebraic mode the command returns NOVAL.

See also: POP, RCLF, STOF

PUT

Type: Command

Description: Put Element Command: Replaces the object at a specified position (second input) in a specified array or list (first input) with a specified object (third input). If the array or list is unnamed, returns the new array or list.

For matrices, $n_{position}$ counts in row order.

Access: [→] [PRG] LIST ELEMENTS PUT ([PRG] is the left-shift of the [EVAL] key).

The spectral radius of a matrix is a measure of the size of the matrix, and is equal to the absolute value of the largest eigenvalue of the matrix.

Access: ← MATRICES OPERATIONS NXT NXT SRAD (MATRICES is the left-shift of the 5 key).
← MTH MATRIX NORMALIZE SRAD (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
⟦ matrix ⟧$_{n\times n}$	→	$x_{spectralradius}$

See also: COND, SNRM, TRACE

SRB

Type: Command

Description: Shift Right Byte Command: Shifts a binary integer one byte to the right.

The least significant byte is shifted out to the right and lost, while the most significant byte is regenerated as zero. SRB is equivalent to binary division by 2^8 (or executing SR eight times).

Access: ← MTH BASE NXT BYTE SRB (MTH is the left-shift of the SYMB key).
→ BASE NXT BYTE SRB (BASE is the right-shift of the 3 key).

Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n$_1$	→	#n$_2$

See also: ASR, SL, SLB, SR

SRECV

Type: Command

Description: Serial Receive Command: Reads up to *n* characters from the serial input buffer and returns them as a string, along with a digit indicating whether errors occurred.

SRECV does not use Kermit protocol.

If *n* characters are not received within the time specified by STIME (default is 10 seconds), SRECV "times out", returning a 0 to level 1 and as many characters as were received to level 2.

If the level 2 output from BUFLEN is used as the input for SRECV, SRECV will not have to wait for more characters to be received. Instead, it returns the characters already in the input buffer.

If you want to accumulate bytes in the input buffer before executing SRECV, you must first open the port using OPENIO (if the port isn't already open).

SRECV can detect three types of error when reading the input buffer:

- Framing errors and UART overruns (both causing "Receive Error" in ERRM).
- Input-buffer overflows (causing "Receive Buffer Overflow" in ERRM).
- Parity errors (causing "Parity Error" in ERRM).

SRECV returns 0 if an error is detected when reading the input buffer, or 1 if no error is detected. Parity errors do not stop data flow into the input buffer. However, if a parity error occurs, SRECV stops reading data after encountering a character with an error.

Framing, overrun, and overflow errors cause all subsequently received characters to be ignored until the error is cleared. SRECV does not detect and clear any of these types of errors until it tries to read the byte where the error occurred. Since these three errors cause the byte where the error occurred and all subsequent bytes to be ignored, the input buffer will be empty after all previously received good bytes have been read. Therefore, SRECV detects and clears these errors when it tries to read a byte from an empty input buffer.

Plot Type	Default Interval
PCONTOUR	RES does not apply
POLAR	2°, 2 grads, or π/90 radians
SCATTER	RES does not apply
SLOPEFIELD	RES does not apply
WIREFRAME	RES does not apply
YSLICE	2 pixels (plots a point in every other column of pixels)

Access: → CAT RES

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n$_{internal}$	→	
#n$_{internal}$	→	

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

RESTORE

Type: Command

Description: Restore HOME Command: Replaces the current *HOME* directory with the specified backup copy (:*n*$_{port}$:*name*$_{backup}$) previously created by ARCHIVE.

The specified port number must be in the range 0 to 3.

To restore a *HOME* directory that was saved on a remote system using :IO:*name* ARCHIVE, put the backup object itself on the stack, execute RESTORE and then execute a warm start.

Access: ← PRG MEM NXT RESTORE (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
:n$_{port}$:name$_{backup}$	→	
backup	→	

Example: To restore a *HOME* directory that was saved as the file *AUG1* on a remote system, execute `'AUG1' SEND` on the remote system, then execute the following on the local calculator:

```
RECV 'AUG1' RCL RESTORE
```

See also: ARCHIVE

RESULTANT

Type: Function

Description: Returns the resultant of two polynomials of the current variable. That is, it returns the determinant of the Sylvester matrix of the two polynomials.

Access: ← ARITH POLY ← PREV

Input: Level 2/Argument 1: The first polynomial.
Level 1/Argument 2: The second polynomial.

Output: The determinant of the two matrices that correspond to the polynomials.

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
n	m	0/1	→	

See also: EQNLIB, MSOLVR

SOLVER

Type: Command

Description: Displays a menu of commands used in solving equations.

Access: [↱] CAT SOLVER

Flags: If the CHOOSE boxes flag is clear (flag –117 clear), displays the operations as a numbered list. If the flag is set, displays the operations as a menu of function keys.

Input/Output: None

SOLVEVX

Type: Command

Description: Finds zeros of an expression with respect to the current variable, or solves an equation with respect to the current variable. (You use the CAS modes input form to set the current variable.)

Access: Symbolic solve, [←] S.SLV , [SYMB] SOLVE

Input: An expression or equation in the current variable. A list of equations and expressions can be given too, each will be solved for the current variable.

Output: A zero or solution, or a list of zeros or solutions.

Flags: Radians mode must be set (flag –17 set).
For a symbolic result, clear the CAS modes numeric option (flag –3 clear).
If Exact mode is set (flag –105 clear) and there are no exact solutions, the command returns a null list even when there are approximate solutions.
If complex mode is set (flag –103 set) then SOLVE will search for complex roots as well as real ones. Complex roots are displayed according to the coordinate system selected.

Example: Solve the following expression for 0, where X is the default variable on the calculator:
$x^3 - x - 9$

Command: `SOLVEVX(X^3-X-9)`

Result: `X=2.24004098747`
Note that if exact mode is set, this example returns a null list as there are no exact solutions to the equation.

See also: LINSOLVE, SOLVE

SORT

Type: Command

Description: Ascending Order Sort Command: Sorts the elements in a list in ascending order.

The elements in the list can be real numbers, strings, lists, names, binary integers, or unit objects. However, all elements in the list must all be of the same type. Strings and names are sorted by character code number. Lists of lists are sorted by the first element in each list.

To sort in reverse order, use SORT REVLIST.

Access: [←] MTH LIST SORT (MTH is the left-shift of the [SYMB] key).
[←] PRG LIST PROCEDURES [NXT] SORT (PRG is the left-shift of the [EVAL] key).

RKF

Type: Command

Description: Solve for Initial Values (Runge–Kutta–Fehlberg) Command: Computes the solution to an initial value problem for a differential equation, using the Runge-Kutta-Fehlberg (4,5) method.

RKF solves y'(t) = f(t,y), where $y(t_0) = y_0$. The arguments and results are as follows:

- { *list* } contains three items in this order: the independent (*t*) and solution (*y*) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).
- x_{tol} sets the absolute error tolerance. If a list is used, the first value is the absolute error tolerance and the second value is the initial candidate step size.
- x_{Tfinal} specifies the final value of the independent variable.

RKF repeatedly calls RKFSTEP as it steps from the initial value to x_{Tfinal}.

Access: [↱] CAT RKF

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_2/I_1	L_1/I_2
{ list }	x_{tol}	$x_{T\,final}$	→	{ list }	x_{tol}
{ list }	{ x_{tol} x_{hstep} }	$x_{T\,final}$	→	{ list }	x_{tol}

L = Level; A = Argument; I = item

Example: Solve the following initial value problem for y(8), given that y(0) = 0:

$$y' = \frac{1}{1+t^2} - 2y^2 = f(t,y)$$

1. Store the independent variable's initial value, 0, in T.
2. Store the dependent variable's initial value, 0, in Y.
3. Store the expression, $\frac{1}{1+t^2} - 2y^2$, in *F*.
4. Enter a list containing these three items: { T Y F }.
5. Enter the tolerance. Use estimated decimal place accuracy as a guideline for choosing a tolerance: 0.00001.
6. Enter the final value for the independent variable: 8.

The stack should look like this:

```
{ T Y F }
.00001
8
```

7. Press RKF. The variable *T* now contains 8, and *Y* now contains the value .123077277659.

The actual answer is .123076923077, so the calculated answer has an error of approximately .00000035, well within the specified tolerance.

See also: RKFERR, RKFSTEP, RRK, RRKSTEP, RSBERR

RKFERR

Type: Command

Description: Error Estimate for Runge–Kutta–Fehlberg Method Command: Returns the absolute error estimate for a given step *h* when solving an initial value problem for a differential equation.

The arguments and results are as follows:

- { *list* } contains three items in this order: the independent (*t*) and solution (*y*) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).
- *h* is a real number that specifies the step.
- y_{delta} displays the change in solution for the specified step.

When plot type is set to SLOPEFIELD, the DRAW command plots a slope representation of a scalar function with two variables. SLOPEFIELD requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

{ 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 SLOPEFIELD, 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 not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

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

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

For plot type SLOPEFIELD, 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 not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command SLOPEFIELD places the command name SLOPEFIELD in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: ↱ CAT SLOPEFIELD

Input/Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, TRUTH, WIREFRAME, YSLICE

SNEG

Type: Command

Description: Store Negate Command: Replaces the contents of a variable with its negative.
The named object must be a number, an array, an algebraic object, a unit object, or a graphics object. For information on negation, see NEG.

Access: ↰ PRG MEMORY ARITHMETIC NXT SNEG (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'name'	→	

See also: NEG, SCONJ, SINV

SNRM

Type: Command

Description: Spectral Norm Command: Returns the spectral norm of an array.
The spectral norm of a vector is its Euclidean length, and is equal to the largest singular value of a matrix.

Access: ↰ MATRICES OPERATIONS NXT NXT SNRM (MATRICES is the left-shift of the 5 key).
↰ MTH MATRIX NORMALIZE SNRM (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n_1	→	#n_2

See also: RLB, RR, RRB

RLB

Type: Command

Description: Rotate Left Byte Command: Rotates a binary integer one byte to the left.
The leftmost byte of #n_1 becomes the rightmost byte of #n_2. RLB is equivalent to executing RL eight times.

Access: ↱ BASE NXT BYTE RLB (BASE is the right-shift of the 3 key).

Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n_1	→	#n_2

See also: RL, RR, RRB

RND

Type: Function

Description: Round Function: Rounds an object to a specified number of decimal places or significant digits, or to fit the current display format.
n_{round} (or $symb_{round}$ if flag –3 is set) controls how the level 2 argument is rounded, as follows:

n_{round} or $symb_{round}$	Effect on Level 2 Argument
0 through 11	Rounded to *n* decimal places.
–1 through –11	Rounded to *n* significant digits.
12	Rounded to the current display format.

For complex numbers and arrays, each real number element is rounded. For unit objects, the numerical part of the object is rounded.

Access: ↰ MTH REAL NXT NXT RND (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
z_1	n_{round}	→	z_2
z	'symb$_{round}$'	→	'RND(symb$_{round}$)'
'symb'	n_{round}	→	'RND(symb,n$_{round}$)'
'symb$_1$'	'symb$_{round}$'	→	'RND('symb$_1$, symb$_{round}$)'
[array$_1$]	n_{round}	→	[array$_2$]
x_unit	n_{round}	→	y_unit
x_unit	'symb$_{round}$'	→	'RND(x_unit, symb$_{round}$)'

Example 1: (4.5792,8.1275) 2 RND returns (4.58,8.13).

Example 2: [2.34907 3.96351 2.73453] -2 RND returns [2.3 4 2.7].

Radians mode must be set (flag –17 set).
Must be in complex mode (flag –103 set).

Example: Express e^{ix} in trigonometric terms.

Command: `SINCOS(EXP(i*X))`

Result: `COS(X)+iSIN(X)`

See also: EXPLN

SINH

Type: Analytic function

Description: Hyperbolic Sine Analytic Function: Returns the hyperbolic sine of the argument.
For complex arguments, $\sinh(x + iy) = \sinh x \cos y + i \cosh x \sin y$.

Access: ▶ TRIG HYPERBOLIC SINH (TRIG is the right-shift of the 8 key).
◀ MTH HYPERBOLIC SINH (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*sinh* z
'symb'	→	'SINH(symb)'

See also: ASINH, COSH, TANH

SINV

Type: Command

Description: Store Inverse Command: Replaces the contents of the named variable with its inverse.
The named object must be a number, a matrix, an algebraic object, or a unit object. For information on reciprocals, see INV.

Access: ◀ PRG MEMORY ARITHMETIC NXT SINV (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'name'	→	

See also: INV, SCONJ, SNEG

SIZE

Type: Command Operation

Description: Size Command: Returns the number of characters in a string, the number of digits in an integer, the number of elements in a list, the dimensions of an array, the number of objects in a unit object or algebraic object, or the dimensions of a graphics object.
The size of a unit is computed as follows: the scalar (+1), the underscore (+1), each unit name (+1), operator or exponent (+1), and each prefix (+2).
Any object type not listed above returns a value of 1.

Access: ◀ PRG NXT CHARS SIZE (PRG is the left-shift of the EVAL key).

ROOT

Type: Command

Description: Root-Finder Command: Returns a real number x_{root} that is a value of the specified variable *global* for which the specified program or algebraic object most nearly evaluates to zero or a local extremum.

ROOT is the programmable form of the HP Solve application.

guess is an initial estimate of the solution. ROOT produces an error if it cannot find a solution, returning the message Bad Guess(es) if one or more of the guesses lie outside the domain of the equation, or returns the message Constant? if the equation returns the same value at every sample point. ROOT does *not* return interpretive messages when a root is found.

Access: ▶ CAT ROOT

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
«program»	'global'	guess	→	x_{root}
«program»	'global'	{ guesses }	→	x_{root}
'symb'	'global'	guess	→	x_{root}
'symb'	'global'	{ guesses }	→	x_{root}

ROT

Type: RPL Command

Description: Rotate Objects Command: Rotates the first three objects on the stack, moving the object on level 3 to level 1.

In RPN mode, ROT is equivalent to 3 ROLL.

Access: ◀ PRG STACK ROT (PRG is the left-shift of the EVAL key).

Input/Output:

L_3	L_2	L_1		L_3	L_2	L_1
obj_3	obj_2	obj_1	→	obj_2	obj_1	obj_3

L = Level

See also: OVER, PICK, ROLL, ROLLD, SWAP, UNROT

ROW–

Type: Command

Description: Delete Row Command: Deletes row *n* of a matrix (or element *n* of a vector), and returns the modified matrix (or vector) and the deleted row (or element).

n_{row} or $n_{element}$ is rounded to the nearest integer.

Access: ◀ MATRICES CRREATE ROW ROW– (MATRICES is the left-shift of the 5 key).
◀ MTH MATRIX ROW ROW– (MTH is the left-shift of the SYMB key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 2/Item 1	Level 1/Item 2
[[matrix]]	n_{row}	→	[[matrix]]	[vector]
[vector]	$n_{element}$	→	[vector]	$element_n$

See also: COL–, COL+, ROW+, RSWP

Access: ◀ MTH REAL ▶ SIGN (MTH is the left-shift of the SYMB key).
▶ CMPLX ▶ SIGN (CMPLX is the right-shift of the 1 key).

Flags: Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z_1	→	z_2
x_unit	→	x_{sign}
'symb'	→	'SIGN(symb)'

Example 1: `32_ft SIGN` returns `1`.

Example 2: `(1,1) SIGN` returns `(.707106781187,.707106781187)`.

See also: ABS, MANT, XPON

SIGNTAB

Type: Command

Description: Tabulates the sign of a rational function of the current CAS variable.

Access: SYMB GRAPH, ◀ CALC GRAPH

Input: An algebraic expression.

Output: A list containing, the points where the expression changes sign, and between each pair of points, the sign of the expression between those points.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Show the ranges of values of x for which the expression $2\text{-}x^2$ is positive and negative.

Command: `SIGNTAB(2 - X^2)`

Result: `{ '-∞' - '-√2' + '√2' - '+∞' }`

See also: TABVAR

SIMP2

Type: Command

Description: Simplifies two objects by dividing them by their greatest common divisor.

Access: Arithmetic, ◀ ARITH ▶

Input: Level 2/Argument 1: The first object.
Level 1/Argument 2: The second object.

Output: Level 2/Item 1: The first object divided by the greatest common divisor.
Level 1/Item 2: The second object divided by the greatest common divisor.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Divide the following expressions by their greatest common divisor:
$x^3 + 6x^2 + 11x + 6$
$x^3 - 7x - 6$

Command: `SIMP2(X^3+6*X^2+11*X+6, X^3-7*X-6)`

Result: `{X+3,X-3}`

RPL>

Type: Command

Description: User RPL program function. This function allows for the entry and execution of User RPL programs while in algebraic mode. While RPL programs can be written in algebraic mode without the use of this function, some RPL constructs, such as FOR…NEXT loops, will produce an error message if not preceded by the RPL> function. As an algebraic function, it will be placed on the command line with a pair of parentheses attached, which must be removed before its use.

For example, to enter the user RPL program of `« 1 5 + »` in algebraic mode, choose the RPL> function from the catalog and press ENTER. Remove the parentheses by pressing ▶ ◀ ◀. Then enter the program by pressing ▶ « » 1 SPC 5 SPC + ENTER. The program object will now be on the first command line. It can be evaluated by pressing EVAL ◀ ANS ENTER.

Access: ▶ CAT RPL>

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	obj

RR

Type: Command

Description: Rotate Right Command: Rotates a binary integer one bit to the right.
The rightmost bit of $\#n_1$ becomes the leftmost bit of $\#n_2$.

Access: ▶ BASE ▶ BIT RR (BASE is the right-shift of the 3 key).
◀ MTH BASE ▶ BIT RR (MTH is the left-shift of the SYMB key).
◀ CONVERT BASE ▶ BIT RR (CONVERT is the left-shift of the 6 key).

Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: RL, RLB, RRB

RRB

Type: Command

Description: Rotate Right Byte Command: Rotates a binary integer one byte to the right.
The rightmost byte of $\#n_1$ becomes the leftmost byte of $\#n_2$. RRB is equivalent to doing RR eight times.

Access: ▶ BASE ▶ BYTE RRB (BASE is the right-shift of the 3 key).
◀ MTH BASE ▶ BYTE RRB (MTH is the left-shift of the SYMB key).
◀ CONVERT BASE ▶ BYTE RRB (CONVERT is the left-shift of the 6 key).

Flags: Binary Integer Wordsize (–5 through –10), Binary Integer Base (–11, –12)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: RL, RLB, RR

User flags are numbered 1 through 128. System flags are numbered –1 through –128. See Appendix C for a listing of system flags and their flag numbers.

Access: ◀ PRG TEST NXT NXT SF (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{flagnumber}$	→	

See also: CF, FC?, FC?C, FS?, FS?C

SHOW

Type: Command

Description: Show Variable Command: Returns $symb_2$, which is equivalent to $symb_1$ except that all implicit references to a variable *name* are made explicit. If the level 1 argument is a list, SHOW evaluates all global variables in $symb_1$ *not* contained in the list.

Access: ◀ CAT SHOW

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'symb$_1$'	'name'	→	'symb$_2$'
'symb$_1$'	{ name$_1$ name$_2$ ... }	→	'symb$_2$'

Example: If 7 is stored in *C* and 5 is stored in *D*:

Command: `'X-Y+2*C+3*D' { X Y } SHOW`

Result: `'X-Y+14+15'`

See also: COLCT, EXPAN, ISOL, QUAD

SIDENS

Type: Function

Description: Silicon Intrinsic Density Command: Calculates the intrinsic density of silicon as a function of temperature, x_T.
If x_T is a unit object, it must reduce to a pure temperature, and the density is returned as a unit object with units of $1/cm^3$.
If x_T is a real number, its units are assumed to be K, and the density is returned as a real number with implied units of $1/cm^3$.
x_T must be between 0 and 1685 K.

Access: ◀ CAT SIDENS

Flags: Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_T	→	$x_{density}$
x_unit	→	x_1/cm^3
'symb'	→	*'SIDENS(*symb*)'*

SIGMA

Type: Function

Description: Calculates the discrete antiderivative of a function *f* with respect to a specified variable. This is a function G such that:

RREFMOD

Type: Command

Description: Performs modular row-reduction to echelon form on a matrix, modulo the current modulus.

Access: Catalog, ◀ CAT

Input: A matrix.

Output: The modular row-reduced matrix. The modulo value is set using the Modes CAS input form.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
If flag –126 is clear (the default), row reduction is done with the last column. If the flag is set, row reduction is done without reducing the last column, but the last column will be modified by the reduction of the rest of the matrix.

Example: Reduce to row-reduced echelon form, modulo 3, the matrix:

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

Command: `rref[[2,1][3,4]]`

Result: `[[-1,0][0,1]]`

See also: rref

RRK

Type: Command

Description: Solve for Initial Values (Rosenbrock, Runge–Kutta) Command: Computes the solution to an initial value problem for a differential equation with known partial derivatives.

RRK solves y'(t) = f(t,y), where $y(t_0) = y_0$. The arguments and results are as follows:

- { *list* } contains five items in this order:
  - The independent variable (*t*).
  - The solution variable (*y*).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the solution variable (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the independent variable (or a variable where the expression is stored).
- x_{tol} sets the tolerance value. If a list is used, the first value is the tolerance and the second value is the initial candidate step size.
- x_{Tfinal} specifies the final value of the independent variable.
  RRK repeatedly calls RKFSTEP as its steps from the initial value to x_{Tfinal}.

Access: ◀ CAT RRK

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_2/I_1	L_1/I_2
{ list }	x_{tol}	$x_{T\,final}$	→	{ list }	x_{tol}
{ list }	{ x_{tol} x_{Istep} }	$x_{T\,final}$		{ list }	x_{tol}

L = Level; A = Argument; I = item

Example: Solve the following initial value problem for y(8), given that y(0) = 0:

$$y' = \frac{1}{1+t^2} - 2y^2 = f(t,y)$$

Data is always sent from a local Kermit, but can be sent either to another local Kermit (which must execute RECV or RECN) or to a server Kermit.

To rename an object when sending it, include the old and new names in an embedded list.

Access: ▶ CAT SEND

Flags: I/O Device flag (–33), I/O Data Format (–35), I/O Messages (–39), I/O Device for Wire (–78)
If flag –35 is clear (ASCII transfer), the translation setting also has an effect.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'name'	→	
{ $name_1$ ... $name_n$ }	→	
{{ $name_{old}$ $name_{new}$ } name ... }	→	

Example 1: Executing `{{ AAA BBB }} SEND` sends the variable named *AAA* but changes its name to *BBB*.

Example 2: Executing `{{ AAA BBB } CCC } SEND` sends *AAA* as *BBB* and sends *CCC* under its own name. (If the new name is not legal on the calculator, just enter it as a string.)

See also: BAUD, CLOSEIO, CKSM, FINISH, KERRM, KGET, PARITY, RECN, RECV, SERVER, TRANSIO

SEQ

Type: Command

Description: Sequential Calculation Command: Returns a list of results generated by repeatedly executing obj_{exec} using *index* over the range x_{start} to x_{end}, in increments of x_{incr}.

obj_{exec} is a program or algebraic object that is a function of *index*. *index* must be a global or local name. The remaining objects can be anything that will evaluate to real numbers.

The action of SEQ for arbitrary inputs can be predicted exactly from this equivalent program.

x_{start} x_{end} FOR *index* obj_{exec} EVAL x_{incr} STEP n → LIST

where n is the number of new objects left on the stack by the FOR … STEP loop. Notice that *index* becomes a local variable regardless of its original type.

Access: ◀ PRG LIST PROCEDURES NXT SEQ (PRG is the left-shift of the EVAL key).

Input/Output:

I. / A.	I. / A.	I. / A.	I. / A.	I. / A.		I. / I.
obj_{exec}	index	x_{start}	x_{end}	x_{incr}	→	{ list }

L = Level; A = Argument; I = item

Example 1: `'n^2' 'n' 1 4 1` returns `{ 1 4 9 16 }`.

Example 2: `« n SQ » 'n' 2 4 1` returns `{ 4 9 16 }`.

See also: DOSUBS, STREAM

SERIES

Type: Command

Description: For a given function, computes Taylor series, asymptotic development and limit at finite or infinite points.

Access: Calculus, SYMB CALC, or Limits and series, ◀ CALC LIMITS & SERIES

Input: Level 3/Argument 1: The function $f(x)$
Level 2/Argument 2: The variable if the limit point is 0, or an equation $x = a$ if the limit point is a. If the function is in terms of the current variable, this can be given as just the value a.

RRKSTEP will use the Euler method to compute the next solution step and will consider the error tolerance satisfied. The Rosenbrock method will fail if the current independent variable is zero and the stepsize ≤ 2.5×10^{-499} or if the variable is nonzero and the stepsize is 2.5×10^{-11} times its magnitude. The Runge–Kutta–Fehlberg method will fail if the current independent variable is zero and the stepsize ≤ 1.3×10^{-498} or if the variable is nonzero and the stepsize is 1.3×10^{-10} times its magnitude.

Access: ▶ CAT RRKS

Input/Output:

L_4/A_1	L_3/A_2	L_2/A_3	L_1/A_4		L_4/I_1	L_3/I_2	L_2/I_3	L_1/I_4
{ list }	x_{ini}	h	last	→	{ list }	x_{ini}	h_{next}	current

L = Level; A = Argument; I = item

See also: RKF, RKFERR, RKFSTEP, RRK, RSBERR

RSBERR

Type: Command

Description: Error Estimate for Rosenbrock Method Command: Returns an error estimate for a given step h when solving an initial values problem for a differential equation.

The arguments and results are as follows:

- { *list* } contains five items in this order:
  - The independent variable (t).
  - The solution variable (y).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the solution variable (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the independent variable (or a variable where the expression is stored).
- h is a real number that specifies the initial step.
- y_{delta} displays the change in solution.
- *error* displays the absolute error for that step. The *absolute* error is the absolute value of the estimated error for a scalar problem, and the row (infinity) norm of the estimated error vector for a vector problem. (The latter is a bound on the maximum error of any component of the solution.) A zero error indicates that the Rosenbrock method failed and Euler's method was used instead.

Access: ▶ CAT RSBER

Input/Output:

L_2/A_1	L_1/A_2		L_4/I_1	L_3/I_2	L_2/I_3	L_1/I_4
{ list }	h	→	{ list }	h	y_{delta}	error

L = Level; A = Argument; I = item

See also: RKF, RKFERR, RKFSTEP, RRK, RRKSTEP

RSD

Type: Command

Description: Residual Command: Computes the residual B – AZ of the arrays B, A, and Z.

A, B, and Z are restricted as follows:

- A must be a matrix.
- The number of columns of A must equal the number of elements of Z if Z is a vector, or the number of rows of Z if Z is a matrix.
- The number of rows of A must equal the number of elements of B if B is a vector, or the number of rows of B if B is a matrix.

- *ptype* is a command name specifying the plot type. Executing the command SCATTER places the name SCATTER in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: ↰ CAT SCATTER

Input/Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

SCHUR

Type: Command

Description: Schur Decomposition of a Square Matrix Command: Returns the Schur decomposition of a square matrix. SCHUR decomposes *A* into two matrices *Q* and *T*:
- If *A* is a complex matrix, *Q* is a unitary matrix, and *T* is an upper-triangular matrix.
- If *A* is a real matrix, *Q* is an orthogonal matrix, and *T* is an upper quasi-triangular matrix (*T* is upper block triangular with 1 × 1 or 2 × 2 diagonal blocks where the 2 × 2 blocks have complex conjugate eigenvalues).

In either case, $A \cong Q \times T \times \mathrm{TRN}(Q)$

Access: ↰ MATRICES FACTORIZATION SCHUR (MATRICES is the left-shift of the 5 key).
↰ MTH MATRIX FACTORS SCHUR (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
[[matrix *]]A*	→	*[[* matrix *]]Q*	*[[* matrix *]]T*

See also: LQ, LU, QR, SVD, SVL, TRN

SCI

Type: Command

Description: Scientific Mode Command: Sets the number display format to scientific mode, which displays one digit to the left of the fraction mark and *n* significant digits to the right.
Scientific mode is equivalent to scientific notation using $n + 1$ significant digits, where $0 \le n \le 11$. (Values for *n* outside this range are rounded to the nearest integer.) In scientific mode, numbers are displayed and printed like this:

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

where the mantissa has the form *n*.(*n* …) and has zero to 11 decimal places, and the exponent has one to three digits.

Access: ↰ & MODE FMT SCI
↰ PRG NXT MODES FMT SCI (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	

Example: The number 103.6 in Scientific mode to four decimal places appears as 1.0360E2.

See also: ENG, FIX, STD

SCLΣ

Type: Command

Description: Scale Sigma Command: Adjusts (x_{min}, y_{min}) and (x_{max}, y_{max}) in *PPAR* so that a subsequent scatter plot exactly fills *PICT*.
When the plot type is SCATTER, the command AUTO incorporates the functions of SCLΣ. In addition, the command SCATRPLOT automatically executes AUTO to achieve the same result. SCLΣ is included for compatibility with the HP 28.

R→C

Type: Command

Description: Real to Complex Command: Combines two real numbers or real arrays into a single complex number or complex array.
The first input represents the real element(s) of the complex result. The second input represents the imaginary element(s) of the complex result.
Array arguments must have the same dimensions.

Access: ↰ PRG TYPE NXT R→C (PRG is the left-shift of the EVAL key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	*(x,y)*
[R-array*1]*	*[* R-array*2]*	→	*[* C-array *]*

See also: C→R, IM, RE

R→D

Type: Function

Description: Radians to Degrees Function: Converts a real number expressed in radians to its equivalent in degrees.
This function operates independently of the angle mode.

Access: ↰ MTH REAL NXT NXT R→D (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*(180/π)*x
'symb'	→	*'R→D(*symb*)'*

See also: D→R

R→I

Type: Function

Description: Converts a real number to an integer.

Access: ↰ CONVERT REWRITE NXT NXT

Flags: Numeric mode must not be set (flag –3 clear).

Input: Level 1/Argument 1: An integral real number or an expression that evaluates to an integral real.

Output: Level 1/Item 1: The real value converted to an integer.

See also: I→R

SAME

Type: Command

Description: Same Object Command: Compares two objects, and returns a true result (1) if they are identical, and a false result (0) if they are not.
SAME is identical in effect to == for all object types except algebraics, names, and some units. (For algebraics and names, == returns an expression that can be evaluated to produce a test result based on numerical values.

Access: ↰ PRG TEST NXT SAME (PRG is the left-shift of the EVAL key).

Input: Level 1/Item 1: The name of a global variable, or a list of global names, to be removed from the REALASSUME list.

Output: Level 1/Item 1: The same name or list of names as was input, even if any of the named variables were not in REALASSUME.

Example: Remove the variables S1 and S2 which are include in the REALASSUME list by default.

Command: `UNASSUME({S1, S2})`

Result: `{S1, S2}`

See also: ADDTOREAL, ASSUME, DEF, LOCAL, UNASSIGN, UNBIND

UNBIND

Type: Command

Description: Removes all local variables created by the LOCAL command, and returns their values. This is useful only if a program needs to remove local variables created earlier in the same program.

Access: Catalog, ▶ CAT

Input: None

Output: Level 1/Item 1: A list of the local variables that have been removed, with their values.

Example: Remove the local variables ←A and ←B created by the example for LOCAL.

Command: `UNBIND`

Result: `{←B=2, ←A=0}`

See also: DEF, LOCAL, STORE, UNASSIGN, UNASSUME

→UNIT

Type: Command

Description: Stack to Unit Object Command: Creates a unit object from a real number and the unit part of a unit object. →UNIT adds units to a real number, combining the number and the unit part of a unit object (the numerical part of the unit object is ignored). →UNIT is the reverse of OBJ→ applied to a unit object.

Access: ▶ CAT →UNIT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y_unit	→	x_unit

See also: →ARRY, →LIST, →STR, →TAG

UNPICK

Type: RPL Command

Description: Replaces the object at level *n*+2 with the object at level 2 and deletes the objects at levels 1 and 2. Can be though of as a "stack poke".

Access: ◀ PRG STACK NXT UNPICK (PRG is the left-shift of the EVAL key).

Input/Output:

L_{n+2}	L_{n+1}	L_3	L_2	L_1		L_n	L_{n-1}	L_1
obj_n	obj_{n-1}	obj_1	obj	n	→	obj	obj_{n-1}	obj_1

Example: Replace the fourth object with an "X":
`55555 4444 333 22 1 "X" 4 UNPICK` returns `55555 "X" 333 22 1`.

See also: OVER, PICK, ROLL, ROLLD, SWAP, ROT

Note that BUFLEN also clears the above-mentioned framing, overrun, and overflow errors. Therefore, SRECV cannot detect an input-buffer overflow after BUFLEN is executed, unless more characters were received after BUFLEN was executed (causing the input buffer to overflow again). SRECV also cannot detect framing and UART overrun errors cleared by BUFLEN. To find where the data error occurred, save the number of characters returned by BUFLEN (which gives the number of "good" characters received), because as soon as the error is cleared, new characters can enter the input buffer.

Access: ▶ CAT SRECV

Flags: I/O Device (–33), I/O Device for Wire (–78)

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
n	→	'string'	*0/1*

Example: If 10 good bytes were received followed by a framing error, then an SRECV command told to read 10 bytes would *not* indicate an error. Only when SRECV tries to read the byte that caused the framing error does it return a 0. Similarly, if the input buffer overflowed, SRECV would not indicate an error until it tried to read the first byte that was lost due to the overflow.

See also: BUFLEN, CLOSEIO, OPENIO, SBRK, STIME, XMIT

SREPL

Type: Command

Description: Find and replace: Finds and replaces a string in a given text object. You supply the following inputs:
Level 3/argument 1: the string to search.
Level 2/argument 2: the string to find.
Level 1/argument 3: the string to replace it with.

Access: ▶ & EVAL NXT SREPL

◀ PRG NXT CHARS NXT SREPL (PRG is the left-shift of the EVAL key).

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
'string'	'string'	'string'	→	'string'

See also: REPL

SST

Type: Operation

Description: Execute Program Step Operation: Returns and executes the next step of a program. If the next step is a subroutine, executes the subroutine in a single step.
SST is not programmable.

Access: ◀ PRG NXT NXT RUN SST (PRG is the left-shift of the EVAL key).

Input/Output: None

See also: NEXT, SST↓

SST↓

Type: Operation

Description: Execute Subroutine Step Operation: Returns and executes the next step of a program or subroutine. If the next step is a subroutine, returns and executes the first step of the subroutine. SST↓ is not programmable.

Access: ◀ PRG NXT NXT RUN SST↓ (PRG is the left-shift of the EVAL key).

Input/Output: None

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'TVM variable'	→	$x_{TVM\ variable}$

See also: AMORT, TVM, TVMBEG, TVMEND

TYPE

Type: Command

Description: Type Command: Returns the type number of an object, as shown in the following table:

Object Type:	Number:
User objects:	
Real number	0
Complex number	1
Character string	2
Real array	3
Complex array	4
List	5
Global name	6
Local name	7
Program	8
Algebraic object	9
Binary integer	10
Graphics object	11
Tagged object	12
Unit object	13
XLIB name	14
Directory object	15
Library	16

Object Type:	Number:
User objects (continued):	
Backup object	17
Real integer	28
Symbolic vector/matrix	29
Built-in Commands:	
Built-in function	18
Built-in command	19
System Objects:	
System binary	20
Extended real	21
Extended complex	22
Linked array	23
Character	24
Code object	25
Library data	26
Mini font	27
Font	30
Extended object	31

Access: ◀ PRG TEST ▶ TYPE (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	n_{type}

See also: SAME, TVARS, VTYPE, ==

UBASE

Type: Function

Description: Convert to SI Base Units Function: Converts a unit object to SI base units.

Access: ◀ CONVERT UNITS TOOLS UBASE (CONVERT is the left-shift of the 6 key).

Flags: Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_unit	→	y_base-units
'symb'	→	*'UBASE(*symb*)'*

Example: `30_knot` `UBASE` returns `15.4333333333_m/s`.

See also: CONVERT, UFACT, →UNIT, UVAL

Access: ▶ CAT STD

Input/Output: None

Example: The following table provides examples of numbers displayed in Standard mode:

Number	Displayed As	Representable With 12 Digits?
10^{11}	`100000000000`	Yes (integer)
10^{12}	`1.E12`	No
10^{-11}	`.000000000001`	Yes
1.2×10^{-11}	`1.23E-11`	No
12.345	`12.345`	Yes

See also: ENG, FIX, SCI

STEP

Type: Command Operation

Description: STEP Command: Defines the increment (step) value, and ends definite loop structure.

See the FOR and START keyword entries for more information.

Access: ◀ PRG BRANCH START/FOR STEP (PRG is the left-shift of the EVAL key).

Input/Output: None

See also: FOR, NEXT, START

STEQ

Type: Command

Description: Store in EQ Command: Stores an object into the reserved variable *EQ* in the current directory.

Access: ▶ CAT STEQ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	

See also: RCEQ

STIME

Type: Command

Description: Serial Time-Out Command: Specifies the period that SRECV (serial reception) and XMIT (serial transmission) wait before timing out.

The value for *x* is interpreted as a positive value from 0 to 25.4 seconds. If no value is given, the default is 10 seconds. If *x* is 0, there is no time-out; that is, the device waits indefinitely, which can drain the batteries.

STIME is not used for Kermit time-out.

Access: ▶ CAT STIME

- *indep* is a name specifying the independent variable on the horizontal axis, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the horizontal plotting range). The default value is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable on the *horizontal* axis, or a binary integer specifying that interval in pixels. The default value is 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 TRUTH places the name TRUTH in *ptype*.
- *depend* is a name specifying the independent variable on the vertical axis, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable on the vertical axis (the vertical plotting range). The default value is *Y*.

The contents of *EQ* must be an expression or program, and cannot be an equation. It is evaluated for each pixel in the plot region. The minimum and maximum values of the independent variables (the plotting ranges) can be specified in *indep* and *depend*; otherwise, the values in (x_{min}, y_{min}) and (x_{max}, y_{max})(the display range) are used. The result of each evaluation must be a real number. If the result is zero, the state of the pixel is unchanged. If the result is nonzero, the pixel is turned on (made dark).

Access: [→] [CAT] TRUTH

Input/Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, WIREFRAME, YSLICE

TSIMP

Type: Command

Description: Performs simplifications on expressions involving exponentials and logarithms. Converts base 10 logarithms to natural logarithms

Access: Exponential and logarithms, [←] EXP&LN [NXT], or [→] TRIG [NXT] [NXT]

Input: An expression

Output: The simplified expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Simplify log(*x*+*x*)

Command: `TSIMP(LOG(X+X))`

Result: `(LN(2)+LN(X))/(LN(5)+LN(2))`

See also: TEXPAND, TLIN

TSTR

Type: Command

Description: Date and Time String Command: Returns a string derived from the date and time.
The string has the form "*DOW DATE TIME*", where *DOW* is a three-letter abbreviation of the day of the week corresponding to the argument *date* and *time*, *DATE* is the argument *date* in the current date format, and *TIME* is the argument *time* in the current time format.

Access: [→] [TIME] TOOLS [NXT] [NXT] TSTR ([TIME] is the right-shift of the [9] key).

Flags: Time Format (–41), Date Format (–42)

n_{index} is a real integer identifying the alarm based on its chronological position in the system alarm list.

Access: [→] [TIME] TOOLS ALRM STOALARM ([TIME] is the right-shift of the [9] key).

Flags: Date Format (–42), Repeat Alarms Not Rescheduled (–43), Acknowledged Alarms Saved (–44)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{time}	→	n_{index}
{ date time }	→	n_{index}
{ date time obj_{action} }	→	n_{index}
{ date time obj_{action} x_{repeat} }	→	n_{index}

Example: With flag –42 clear, this command:

```
{ 11.06 15.2530 RUN 491520 } STOALARM
```

sets a repeating alarm for November 6 of the currently specified year, at 3:25:30 pm. The alarm action is to execute variable *RUN*. The repeat interval is 491520 clock ticks (1 minute).

See also: DELALARM, FINDALARM, RCLALARM

STOF

Type: Command

Description: Store Flags Command: Sets the states of the system flags or the system and user flags.

With argument #n_{system}, STOF sets the states of the system flags (–1 through –128) only. With argument { #n_{system}, #n_{user}, #$n_{system2}$ #n_{user2} }, STOF sets the states of both the system and user flags.

A bit with value 1 sets the corresponding flag; a bit with value 0 clears the corresponding flag. The rightmost (least significant) bit of #n_{system} and #n_{user} correspond to the states of system flag –1 and user flag +1, respectively.

STOF can preserve the states of flags before a program executes and changes the states. RCLF can then recall the flag's states after the program is executed.

Access: [→] [CAT] STOF

Flags: Binary Integer Wordsize (–5 through –10)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n_{system}	→	
{ #n_{system} #n_{user} #$n_{system2}$ #n_{user2} }	→	

See also: RCLF, PUSH, POP, STWS, RCWS

STOKEYS

Type: Command

Description: Store Key Assignments Command: Defines multiple keys on the user keyboard by assigning objects to specified keys.

x_{key} is a real number of the form *r*.*p* specifying the key by its row number *r*, its column number *c*, and its plane (shift) *p*. (For a definition of plane, see the entry for ASN).

The optional initial list parameter or argument S restores all keys without user assignments to their *standard* key assignments on the user keyboard. This is meaningful only when all standard key assignments had been suppressed (for the user keyboard) by the command S DELKEYS.

If the argument *obj* is the name SKEY, the specified key is restored to its *standard key* assignment.

Access: [→] [CAT] STOKEYS

Flags: User-Mode Lock (–61) and User Mode (–62) affect the status of the user keyboard

Access: Trigonometry, ↱ TRIG NXT NXT

Input: An expression with trigonometric terms.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Express the following in tan terms:
$(\sin x)^2$

Command: `TRIGTAN(SIN(X)^2)`

Result: `TAN(X)^2/(TAN(X)^2+1)`

See also: TRIG, TRIGCOS, TRIGSIN

TRN

Type: Command

Description: Transpose Matrix Command: Returns the (conjugate) transpose of a matrix.
TRN replaces an $n \times m$ matrix A with an $m \times n$ matrix A^T, where:
AijT = Aji for real matrices and AijT = CONJ(Aji) for complex matrices
If the matrix is specified by *name*, $\mathbf{A}^T$ replaces **A** in *name*.

Access: ↰ MTH MATRIX MAKE TRN (MTH is the left-shift of the SYMB key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[matrix *]]*	→	*[[* matrix *]]transpose*
'name'	→	

Example: `[[ 2 3 1 ][ 4 6 9]] TRN` returns `[[ 2 4 ][ 3 6 ][ 1 9 ]]`.

See also: CONJ, TRAN

TRNC

Type: Function

Description: Truncate Function: Truncates an object to a specified number of decimal places or significant digits, or to fit the current display format.
$n_{truncate}$ (or $symb_{truncate}$ if flag –3 is set) controls how the level 2 argument is truncated, as follows:

$n_{truncate}$	Effect on Level 2 Argument
0 through 11	truncated to *n* decimal places
–1 through –11	truncated to *n* significant digits
12	truncated to the current display format

For complex numbers and arrays, each real number element is truncated. For unit objects, the number part of the object is truncated.

Access: ↰ MTH REAL NXT NXT TRNC (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'name'	→	
'name'	obj	→	

See also: STO–, STO*, STO/, +

STO–

Type: Command

Description: Store Minus Command: Calculates the difference between a number (or other object) and the contents of a specified variable, and stores the new value in the specified variable.
The object on the stack and the object in the variable must be suitable for subtraction with each other. STO– can subtract any combination of objects suitable for stack subtraction.
Using STO– to subtract two arrays (where *obj* is an array and *name* is the global name of an array) requires less memory than using the stack to subtract them.

Access: ↰ PRG MEMORY ARITHMETIC STO– (PRG is the left-shift of the EVAL key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'name'	→	
'name'	obj	→	

See also: STO+, STO*, STO/, –

STO*

Type: Command

Description: Store Times Command: Multiplies the contents of a specified variable by a number or other object.
The object on the stack and the object in the variable must be suitable for multiplication with each other. When multiplying two arrays, the result depends on the order of the arguments. The new object of the named variable is the level 2 array times the level 1 array. The arrays must be conformable for multiplication.
Using STO* to multiply two arrays or to multiply a number and an array (where *obj* is an array or a number and *name* is the global name of an array) requires less memory than using the stack to multiply them.

Access: ↰ PRG MEMORY ARITHMETIC STO* (PRG is the left-shift of the EVAL key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'name'	→	
'name'	obj	→	

See also: STO+, STO–, STO/, *

STO/

Type: Command

Description: Store Divide Command: Calculates the quotient of a number (or other object) and the contents of a specified variable, and stores the new value in the specified variable.
The new object of the specified variable is the level 2 object divided by the level 1 object.
The object on the stack and the object in the variable must be suitable for division with each other. If both objects are arrays, the divisor (level 1) must be a square matrix, and the dividend (level 2) must have the same number of columns as the divisor.

Access: ↰ MATRICES OPERATIONS NXT NXT TRAN (MATRICES is the left-shift of the 5 key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[matrix *]]*	→	*[[* matrix *]]*$_{transpose}$
'name'	→	

See also: CONJ, TRN

TRANSIO

Type: Command

Description: I/O Translation Command: Specifies the character translation option. These translations affect only ASCII Kermit transfers and files printed to the serial port.

Legal values for *n* are as follows:

n	Effect
0	No translation
1	Translate character 10 (line feed only) to /from characters 10 and 13 (line feed with carriage return, the Kermit protocol) (the default value)
2	Translate characters 128 through 159 (80 through 9F hexadecimal)
3	Translate all characters (128 through 255)

Access: ↱ CAT TRANSIO

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n$_{option}$	→	

See also: BAUD, CKSM, PARITY

TRIG

Type: Command

Description: Converts complex logarithmic and exponential subexpressions into their equivalent trigonometric expressions. It also simplifies trigonometric expressions by using: $(\sin x)^2 + (\cos x)^2 = 1$

Access: SYMB TRIG, Trigonometry, ↱ TRIG NXT NXT

Input: A complex expression with logarithmic and/or exponential terms, or a trigonometric expression.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
Prefers cosine terms if "prefer cos" is selected (flag –116 clear), prefers sine terms if flag –116 is set.
Must be in Complex mode (flag –103 set) if a complex expression is being simplified.

Example: Express the following in trigonometric terms:
$\ln(x + i)$

Command: `TRIG(LN(X+i))`

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	"obj"

Example: →STR can create special displays to label program output or provide prompts for input. The sequence
`"Result = " SWAP →STR + 1 DISP 1 FREEZE`
displays `Result =` *object* in line 1 of the display, where *object* is a string form of an object taken from level 1.

See also: →ARRY, →LIST, STR→, →TAG, →UNIT

STREAM

Type: Command

Description: Stream Execution Command: Moves the first two elements from the list onto the stack, and executes *obj*. Then moves the next element (if any) onto the stack, and executes *obj* again using the previous result and the new element. Repeats this until the list is exhausted, and returns the final result.
STREAM is nominally designed for *obj* to be a program or command that requires two arguments and returns one result.

Access: ↰ PRG LIST PROCEDURES STREAM (PRG is the left-shift of the EVAL key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ list }	obj	→	result

Example 1: `{ 1 2 3 4 5 } « * » STREAM` returns 120.

Example 2: `« + » STREAM` is equivalent to ΣLIST.

See also: DOSUBS

STRM

Type: Command

Description: Starts the StreamSmart application.

Access: APPS STREAMSMART

Input/Output: None

STURM

Type: Command

Description: For a polynomial *P*, STURM returns a list containing Sturm's sequences of *P* and their multiplicities

Access: Arithmetic, ↰ ARITH POLYNOMIAL ↰ PREV

Input: A polynomial *P*

Output: A list containing the Sturm's sequences for *P*, and the multiplicity for each (as a real number).

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Find the Sturm sequences and their multiplicities for the polynomial:
$x^3 + 1$

Command: `STURM(X^3+1)`

Result: `{[1],-1.,[1],1.,[X^3+1,-(3*X^2),-1],1.}`

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$x_{initial}$	y_{delta}	→	x_{final}
x_unit1	y_unit2$_{delta}$	→	x_unit1$_{final}$
x_unit	'symb'	→	'TINC(x_unit, symb)'
'symb'	y_unit$_{delta}$	→	'TINC(symb, y_unit$_{delta}$)'
'symb$_1$'	'symb$_2$'	→	'TINC(symb$_1$, symb$_2$)'

See also: TDELTA

TLIN

Type: Command

Description: Linearizes and simplifies trigonometric expressions. Note that this function does not collect sin and cos terms of the same angle.

Access: SYMB TRIG, Trigonometry, ↱ TRIG NXT

Input: An expression.

Output: The transformation of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Linearize and simplify the following:
$(\cos(x))^4$

Command: `TLIN(COS(X)^4)`

Result: `(1/8)*COS(4X)+(1/2)*COS(2X)+(3/8)`

See also: SIMPLIFY, TCOLLECT, TEXPAND

TLINE

Type: Command

Description: Toggle Line Command: For each pixel along the line in *PICT* defined by the specified coordinates, TLINE turns off every pixel that is on, and turns on every pixel that is off.

Access: ↰ PRG NXT PICT TLINE (PRG is the left-shift of the EVAL key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
(x$_1$,y$_1$)	(x$_2$,y$_2$)	→	
{ #n$_1$ #m$_1$ }	{ #n$_2$ #m$_2$ }	→	

Example: The following program toggles on and off 10 times the pixels on the line defined by user-unit coordinates (1,1) and (9,9). Each state is maintained for .25 seconds.

```
« ERASE 0 10 XRNG 0 10 YRNG
 { # 0d # 0d } PVIEW
 « 1 10 START
  (1,1) (9,9) TLINE .25 WAIT
  NEXT
 »
»
```

See also: ARC, BOX, LINE

Access: ↰ PRG LIST SUB (PRG is the left-shift of the EVAL key).

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[[matrix]]$_1$	n$_{startposition}$	n$_{endposition}$	→	[[matrix]]$_2$
[[matrix]]$_1$	{n$_{row}$, n$_{column}$ }	n$_{endposition}$	→	[[matrix]]$_2$
[[matrix]]$_1$	n$_{startposition}$	{n$_{row}$, n$_{column}$ }	→	[[matrix]]$_2$
[[matrix]]$_1$	{n$_{row}$, n$_{column}$ }	{n$_{row}$, n$_{column}$ }	→	[[matrix]]$_2$
"string$_{target}$"	n$_{startposition}$	n$_{endposition}$	→	"string$_{result}$"
{ list$_{target}$ }	n$_{startposition}$	n$_{endposition}$	→	{ list$_{result}$ }
grob$_{target}$	{ #n$_1$, #m$_1$ }	{ #n$_2$ #m$_2$ }	→	grob$_{result}$
grob$_{target}$	(x$_1$, y$_1$)	(x$_2$, y$_2$)	→	grob$_{result}$
PICT	{ #n$_1$, #m$_1$ }	{ #n$_2$ #m$_2$ }	→	grob$_{result}$
PICT	(x$_1$, y$_1$)	(x$_2$, y$_2$)	→	grob$_{result}$

Example 1: `{ A B C D E } 2 4 SUB` returns `{ B C D }`.
Example 2: `"ABCDE" 0 10 SUB` returns `"ABCDE"`.
Example 3: `PICT { # 10d #20d } { # 20d # 40d } SUB` returns `Graphic 11 × 21`.

See also: CHR, GOR, GXOR, NUM, POS, REPL, SIZE

SUBST

Type: Function

Description: Substitutes a value for a variable in an expression. The value can be numeric or an expression. This is similar to the Where function, denoted by the symbol |, but SUBST substitutes without evaluating the resulting expression.

Access: Algebra, ↱ ALG NXT, SYMB ALG

Input: Level 2/Argument 1: An expression.
Level 1/Argument 2: The value or expression to be substituted.

Output: The expression with the substitution made.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Substitute $x = z+1$ for x in the following expression, and apply the EXPAND command to simplify the result:
$x^2 + 3x + 7$

Command:
```
SUBST(X^2+3*X+7,X=Z+1)
EXPAND(ANS(1))
```

Result: `Z^2+5*Z+11`

See also: | (where command)

SUBTMOD

Type: Function

Description: Performs a subtraction, modulo the current modulus.

Access: Arithmetic, ↰ ARITH MODULO NXT

TEVAL

Type: Function

Description: For the specified operation, performs the same function as EVAL, and returns the time taken to perform the evaluation as well as the result.

Access: ↱ CAT TEVAL

Input/Output:

Level 1/Argument 1		Level 2/Item 2	Level 1/Item 1
Object	→	result	time taken

See also: EVAL

TEXPAND

Type: Command

Description: Expands transcendental functions.

Access: ↰ EXP&LN, SYMB ALG, SYMB TRIG, ↱ ALG NXT, ↱ TRIG NXT, SYMB NXT EXPLN

Input: An expression.

Output: The transformation of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Expand the following expression:
ln(sin(*x*+*y*))

Command: `TEXPAND(LN(SIN(X+Y)))`

Result: `LN(COS(Y)*SIN(X)+SIN(Y)* COS(X))`

See also: TCOLLECT, TLIN

TEXT

Type: Command

Description: Show Stack Display Command: Displays the stack display.

TEXT switches from the graphics display to the stack display. TEXT does not update the stack display.

Access: ↰ PRG NXT OUT TEXT (PRG is the left-shift of the EVAL key).

Input/Output: None

Example: The command sequence `DRAW 5 WAIT TEXT` selects the graphics display and plots the contents of the reserved variable *EQ* (or reserved variable *ΣDAT*). It subsequently waits for 5 seconds, and then switches back from the graphics display to the stack display.

See also: PICTURE, PVIEW

THEN

Type: Command

Description: THEN Command: Starts the true-clause in conditional or error-trapping structure.
See the IF and IFFER entries for more information.

Access: ↰ PRG BRANCH IF/CASE THEN (PRG is the left-shift of the EVAL key).

Input/Output: None

See also: CASE, ELSE, END, IF IFERR

See also: DUP, DUPN, DUP2, OVER, PICK, ROLL, ROLLD, ROT

SYSEVAL

Type: Command

Description: Evaluate System Object Command: Evaluates unnamed operating system objects specified by their memory addresses.

WARNING: Use extreme care when executing this function. Using SYSEVAL with random addresses will almost always cause a memory loss. Do not use this function unless you know what you are doing.

Access: ↱ CAT SYSEVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_{address}$	→	

Example: Display the version string of a calculator by executing `#2F389h SYSEVAL`. This should display `"HPHP49-C"`.

See also: EVAL, LIBEVAL, FLASHEVAL

SYLVESTER

Type: Command

Description: For a symmetric matrix A, returns D and P where D is a diagonal matrix and
$A = P^T D P$

Access: ↰ MATRICES QUADF

Input: A symmetric matrix.

Output: Level 2/Item 1: the diagonal matrix, D.
Level 1/Item 2: The matrix P.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Rewrite in $P^T D P$ form the matrix:

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

Command: `SYLVESTER([1, 2][2, 4])`

Result: `{[1, 0], [[1, 2][0, 1]]}`

SYST2MAT

Type: Command

Description: Converts a system of linear equations in algebraic form to matrix form.

Access: ↰ CONVERT MATRX, Matrices, ↰ MATRICES LINEAR SYSTEMS

Input: Level 2/Argument 1: A vector containing a system of linear equations. An expression with no equal sign is treated as an equation setting the expression equal to zero.
Level 1/Argument 2: A vector whose elements are the system's variables. The variables must not exist in the current path.

Output: A matrix that represents the system of linear equations.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

TAYLOR0

Type: Function

Description: Performs a fourth-order Taylor expansion of an expression at $x = 0$.

Access: Calculus, ↰ CALC LIMITS & SERIES, SYMB CALC NXT

Input: An expression

Output: The Taylor expansion of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Obtain the fourth-order Taylor series expansion of cos(x) at x=0.

Command: `TAYLR0(COS(X))`

Result: `1/24*X^4+-1/2*X^2+1`

See also: DIVPC, lim, TAYLR, SERIES

TAYLR

Type: Command

Description: Taylor Polynomial Command: Calculates the *n*th order Taylor polynomial of *symb* in the variable *global*.
The polynomial is calculated at the point *global* = 0. The expression *symb* may have a removable singularity at 0. The order, *n*, is the *relative* order of the Taylor polynomial — the difference in order between the largest and smallest power of *global* in the polynomial.
TAYLR always returns a symbolic result, regardless of the state of the Numeric Results flag (–3).

Access: ↰ CALC LIMITS & SERIES TAYLR (CALC is the left-shift of the 4 key).

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
'symb'	'global'	n_{order}	→	$'symb_{Taylor}'$

Example: The command sequence `'1+SIN(X)^2' 'X' 5 TAYLR` returns `'1+X^2-8/4!*X^4'`.

See also: ∂, ∫, Σ

TCHEBYCHEFF

Type: Function

Description: Returns the nth Tchebycheff polynomial.

Access: Catalog, ↱ CAT

Input: A non-negative integer, *n*.

Output: The *n*th Tchebycheff polynomial.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Obtain the fourth Tchebycheff polynomial.

Command: `TCHEBYCHEFF(4)`

Result: `8*X^4-8*X^2+1`

See also: HERMITE, LEGENDRE

Result: `{ X^2+1,{{1, 2, 3},{2, 5, 10}}}`

TABVAR

Type: Command

Description: For a function of the current variable, with a rational derivative, computes the variation table, that is the turning points of the function and where the function is increasing or decreasing.

Access: SYMB GRAPH NXT, ↰ CALC GRAPH NXT

Input: An expression in terms of the current variable, which has a rational derivative.

Output: Level 3/Item 1: The original rational function.
Level 2/Item 2: A list of two lists. The first list indicates the variation of the function (where it is increasing or decreasing) in terms of the independent variable. The second list indicates the variation in terms of the dependent variable, the function value.
Level 1/Item 3: A graphic object that shows how the variation table was computed.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).

Example: Tabulate the variation of the function:
$x^2 - 1$

Command: `TABVAR(X^2-1)`

Result: `{'X^2-1' {{ '-∞' - 0 + '∞' }{ '+∞' ↓ '-1' ↑ '+∞' }} Graphic 96 × 55 }`
Viewing the graphic, one sees the original function F and its derivative, as functions of X, and the variation table for X and F, shown as a matrix

See also: SIGNTAB

→TAG

Type: Command

Description: Stack to Tag Command: Combines objects in levels 1 and 2 to create tagged (labeled) object.
The "*tag*" argument is a string of fewer than 256 characters.

Access: ↰ PRG TYPE →TAG (PRG is the left-shift of the EVAL key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	"tag"	→	:tag:obj
obj	'name'	→	:name:obj
obj	x	→	:x:obj

See also: →ARRY, DTAG, →LIST, OBJ→, →STR, →UNIT

TAIL

Type: Command

Description: Last Listed Elements Command: Returns all but the first element of a list or string.

Access: ↰ PRG NXT CHARS NXT TAIL (PRG is the left-shift of the EVAL key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ obj_1 ... obj_n }	→	{ obj_2 ... obj_n }
"$string_1$"	→	"$string_2$"

Example: `"tail" TAIL` returns `"ail"`.

example, that "a" is greater than or equal to "B", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ◀ ≥ (≥ is the left-shift of the 1/x key).

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	*0/1*
#n$_1$	#n$_2$	→	*0/1*
"string$_1$"	"string$_2$"	→	*0/1*
x	'symb'	→	'x ≥ symb'
'symb'	x	→	'symb ≥ x'
'symb$_1$'	'symb$_2$'	→	'symb$_1$ ≥ symb$_2$'
x_unit$_1$	y_unit$_2$	→	*0/1*
x_unit	'symb'	→	'x_unit ≥ symb'
'symb'	x_unit	→	'symb ≥ x_unit'

See also: <, ≤, >, ==, ≠

≠ (Not equal)

Type: Function

Description: Not Equal Function: Tests if two objects are not equal.

The function ≠ returns a true result (1) if the two objects have different values, or a false result (0) otherwise. (Lists and programs are considered to have the same values if the objects they contain are identical.)

If one object is algebraic or a name, and the other is a number, a name, or algebraic, ≠ returns a symbolic comparison expression that can be evaluated to return a test result.

If the imaginary part of a complex number is 0, it is ignored when the complex number is compared to a real number, so, for example, 6 and (6,0) and considered to be equal.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ◀ ≠ (≠ is the left-shift of the +/- key).

Flags: Numerical Results (–3)

UNROT

Type: RPL Command

Description: Changes the order of the first three objects on the stack. The order of the change is the opposite to that of the ROT command.

Access: ◀ PRG STACK UNROT (PRG is the left-shift of the EVAL key).

Input/Output:

L$_3$	L$_2$	L$_1$		L$_3$	L$_2$	L$_1$
obj$_3$	obj$_2$	obj$_1$	→	obj$_1$	obj$_3$	obj$_2$

Example: 333 22 1 UNROT returns 1 333 22.

See also: OVER, PICK, ROLL, ROLLD, SWAP, ROT

UNTIL

Type: Command

Description: UNTIL Command: Starts the test clause in a DO … UNTIL … END indefinite loop structure. See the DO entry for more information.

Access: ◀ PRG BRANCH DO UNTIL (PRG is the left-shift of the EVAL key).

Input/Output: None

See also: DO, END

UPDIR

Type: Command

Description: Up Directory Command: Makes the parent of the current directory the new current directory. UPDIR has no effect if the current directory is *HOME*.

Access: ◀ UPDIR (UPDIR is the left-shift of the VAR key).

Input/Output: None

See also: CRDIR, HOME, PATH, PGDIR

UTPC

Type: Command

Description: Upper Chi-Square Distribution Command: Returns the probability utpc(n, x) that a chi-square random variable is greater than x, where n is the number of degrees of freedom of the distribution.

The defining equations are these:

- For x ≥ 0:

$$utpc(n,x) = \left[\frac{1}{2^{\frac{n}{2}}\Gamma\left(\frac{n}{2}\right)}\right]\int_x^\infty t^{\frac{n}{2}-1}\cdot e^{-\frac{t}{2}}dt$$

- For x < 0:

$$utpc(n,x)) = 1$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the factorial command.

The value n is rounded to the nearest integer and, when rounded, must be positive.

Access: ◀ MTH NXT PROBABILITY NXT UTPC (MTH is the left-shift of the SYMB key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	*0/1*
#n$_1$	#n$_2$	→	*0/1*
"string$_1$"	"string$_2$"	→	*0/1*
x	'symb'	→	'x < symb'
'symb'	x	→	'symb < x'
'symb$_1$'	'symb$_2$'	→	'symb$_1$ < symb$_2$'
x_unit$_1$	y_unit$_2$	→	*0/1*
x_unit	'symb'	→	'x_unit < symb'
'symb'	x_unit	→	'symb < x_unit'

See also: ≤, >, ≥, ==, ≠

≤ (Less than or Equal)

Type: Function

Description: Less Than or Equal Function: Tests whether one object is less than or equal to another object.

The function ≤ returns a true test result (1) if the first argument is less than or equal to the second argument, or a false test result (0) otherwise. If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, ≤ returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "less than or equal" means numerically equal or smaller (1 is less than 2). For real numbers, "less than or equal" also means equally or more negative (–2 is less than –1). For strings, "less than or equal" means alphabetically equal or previous ("ABC" is less than or equal to "DEF"; "AAA" is less than or equal to "AAB"; "A" is less than or equal to "AA"). In general, characters are ordered according to their character codes. This means, for example, that "B" is less than "a", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperature and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ◀ ≤ (≤ is the left-shift of the X key above the 8).

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	*0/1*
#n$_1$	#n$_2$	→	*0/1*
"string$_1$"	"string$_2$"	→	*0/1*
x	'symb'	→	'x ≤symb'
'symb'	x	→	'symb ≤x'
'symb$_1$'	'symb$_2$'	→	'symb$_1$ ≤symb$_2$'
x_unit$_1$	y_unit$_2$	→	*0/1*
x_unit	'symb'	→	'x_unit ≤symb'
'symb'	x_unit	→	'symb ≤x_unit'

See also: <, >, ≥, ==, ≠

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
m	v	x	→	utpn(m,v,x)

See also: UTPC, UTPF, UTPT

UTPT

Type: Command

Description: Upper Student's t Distribution Command: Returns the probability utpt(*n*, *x*) that a Student's *t* random variable is greater than *x*, where *n* is the number of degrees of freedom of the distribution.

The following is the defining equation for all *x*:

$$utpt(n,x) = \left[\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\right]\int_x^\infty \left(1+\frac{t^2}{n}\right)^{-\frac{n+1}{2}} dt$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the factorial command.

The value *n* is rounded to the nearest integer and, when rounded, must be positive.

Access: ◀ MTH (NXT) PROBABILITY (NXT) UTPT (MTH is the left-shift of the SYMB key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	x	→	utpt(n,x)

See also: UTPC, UTPF, UTPN

UVAL

Type: Function

Description: Unit Value Function: Returns the numerical part of a unit object.

Access: ▶ UNITS TOOLS UVAL (UNITS is the right-shift of the 6 key).

Flags: Numerical Results (–3)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_unit	→	x
'symb'	→	'UVAL(symb)'

See also: CONVERT, UBASE, UFACT, →UNIT

V→

Type: Command

Description: Vector/Complex Number to Stack Command: Separates a vector or complex number into its component elements.

For vectors with four or more elements, V→ executes *independently* of the coordinate system mode, and always returns the elements of the vector to the stack as they are stored internally (in rectangular form). Thus, V→ is equivalent to OBJ→ for vectors with four or more elements.

Access: ◀ MTH VECTOR V→ (MTH is the left-shift of the SYMB key).

Flags: Coordinate System (–15 and –16)

$$\Gamma(x+1) = \int_0^\infty e^{-t} t^x \, dt$$

and defined for other values of x by analytic continuation: $\Gamma(x+1) = n\,\Gamma(x)$

Access: ◀┐ MTH NXT PROBABILITY ! (MTH is the left-shift of the SYMB key).

Flags: Numerical Results (–3), Underflow Exception (–20), Overflow Exception (–21)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	n!
x	→	*Γ(x + 1)*
'symb'	→	*'(symb!)'*

See also: COMB, PERM

% (Percent)

Type: Function

Description: Percent Function: Returns *x* percent of *y*.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C=273.15 K, and 0 °F=459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C=1 K and 1 °F=1 °R.

The arithmetic operators +, –, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, –, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

For more information on using temperature units with arithmetic functions, see the entry for +.

Access: ◀┐ MTH REAL % (MTH is the left-shift of the SYMB key).

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	xy/*100*
x	'symb'	→	*'%(*x,symb*)'*
'symb'	x	→	*'%(*symb,x*)'*
'symb$_1$'	'symb$_2$'	→	*'%(*symb$_1$, symb$_2$*)'*
x	y_unit	→	*(*xy/*100)*_unit
x_unit	y	→	*(*xy/*100)*_unit
'symb'	x_unit	→	*'%(*symb,x_unit*)'*
x_unit	'symb'	→	*'%(*x_unit,symb*)'*

Example 1: `23.7 995 %` returns `235.815`.

Example 2: `15 176_kg %` returns `26.4_kg`.

Example 3: `100_°C 50 %` returns `50_°C`.

See also: +, %CH, %T

_ (Unit attachment)

Type: Unit attachment

Description: Unit attachment symbol: Attaches a unit type to a numeric value.

Mode	Result
Rectangular (flag –16 clear)	[x$_1$ x$_2$ x$_3$]
Polar/Cylindrical (flag –15 clear and –16 set)	[x$_1$ x∠$_{theta}$ x$_z$]
Polar/Spherical (flag –15 and –16 set)	[x$_1$ x∠$_{theta}$ x∠$_{phi}$]

Access: ◀┐ MTH VECTOR →V3 (MTH is the left-shift of the SYMB key).

Flags: Coordinate System (–15 and –16)

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
x$_1$	x$_2$	x$_3$	→	[x$_1$ x$_2$ x$_3$]
x$_1$	x$_{theta}$	x$_z$	→	[x$_1$ ∠ x$_{theta}$ x$_z$]
x$_1$	x$_{theta}$	x$_{phi}$	→	[x$_1$ ∠ x$_{theta}$ x$_{phi}$]

Example 1: With flag –16 clear (Rectangular mode), `1 2 3 →V3` returns `[ 1 2 3 ]`.

Example 2: With flag –15 clear and –16 set (Polar/Cylindrical mode), `1 2 3 →V3` returns `[ 1 ∠2 3 ]`.

Example 3: With flags –15 and –16 set (Polar/Spherical mode), `1 2 3 →V3` returns `[ 1 ∠2 ∠3 ]`.

See also: V→, →V2

VANDERMONDE

Type: Command

Description: Builds the Vandermonde matrix (also called the alternant matrix) from a list of objects. That is, for a list of n objects, the command creates an $n \times n$ matrix. The i^{th} column in the matrix consists of the list items raised to the power of $(i–1)$. Sometimes the Vandermonde matrix is defined with the i^{th} row containing the items raised to the power of $(i–1)$; to obtain this, transpose the result with the command TRAN.

Access: Matrices, ◀┐ MATRICES CREATE NXT NXT, ◀┐ MTH MATRX MAKE NXT NXT

Input: A list of objects. A vector is allowed too.

Output: The corresponding Vandermonde matrix.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).

Example: Build the row version of the Vandermonde matrix from the following list of objects:
$\{x, y, z\}$

Command: `TRAN(VANDERMONDE({x,y,z}))`

Result:

$$\begin{bmatrix} 1 & 1 & 1 \\ x & y & z \\ x^2 & y^2 & z^2 \end{bmatrix}$$

See also: CON, HILBERT, IDN, RANM

VAR

Type: Command

Description: Variance Command: Calculates the sample variance of the coordinate values in each of the m columns in the current statistics matrix (ΣDAT).

- To enter one data point with a single coordinate value, the argument for Σ+ must be a real number.
- To enter one data point with multiple coordinate values, the argument for Σ+ must be a vector with *m* real coordinate values.
- To enter several data points, the argument for Σ+ must be a matrix of *n* rows of *m* real coordinate values.

In each case, the coordinate values of the data point(s) are added as new rows to the current statistics matrix (reserved variable *ΣDAT*). If *ΣDAT* does not exist, Σ+ creates an *n* x *m* matrix and stores the matrix in *ΣDAT*. If *ΣDAT* does exist, an error occurs if it does not contain a real matrix, or if the number of coordinate values in each data point entered with Σ+ does not match the number of columns in the current statistics matrix.

Once *ΣDAT* exists, individual data points of *m* coordinates can be entered as *m* separate real numbers or an *m*-element vector. LASTARG returns the *m*-element vector in either case.

Access: [→] CAT Σ+

Input/Output:

$L_m/A_1 \ldots L_2/A_{m-1}$	L_1/A_m		L_1/I_1
	x	→	
	$[\,x_1, x_2, \ldots, x_m\,]$	→	
	$[[\,x_{1\,1}, \ldots, x_{1\,m}\,]\,[\,x_{n\,1}, \ldots, x_{n\,m}\,]]$	→	
$x_1 \ldots x_{m-1}$	x_m	→	

L = Level; A = Argument; I = Item

Example: The sequence `CLΣ [ 2 3 4 ] Σ+ 3 1 7 Σ+` creates the matrix `[[ 2 3 4 ][ 3 1 7 ]]` in *ΣDAT*.

See also: CLΣ, RCLΣ, STOΣ, Σ–

Σ– (Sigma Minus)

Type: Command

Description: Sigma Minus Command: Returns a vector of *m* real numbers (or one number *x* if *m* = 1) corresponding to the coordinate values of the last data point entered by Σ+ into the current statistics matrix (reserved variable *ΣDAT*).

The last row of the statistics matrix is deleted.

The vector returned by Σ– can be edited or replaced, then restored to the statistics matrix by Σ+.

Access: [→] CAT Σ–

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x
	→	$[\,x_1\ x_2 \ldots x_m\,]$

See also: CLΣ, RCLΣ, STOΣ, Σ+

π (Pi)

Type: Function

Description: π Function: Returns the symbolic constant ' π ' or its numerical representation, 3.14159265359.

The number returned for π is the closest approximation of the constant π to 12-digit accuracy.

In Radians mode with flag –2 and –3 clear (to return symbolic results), trigonometric functions of π and π/2 are automatically simplified. For example, evaluating 'SIN(π)' returns zero. However, if

VISIT

Type: Command

Description: For a specified variable, opens the contents in the command-line editor.

Access: [→] CAT VISIT or [←] [▼]

Input/Output:

Level 1/Argument 1		Level 1/Item 1
A variable name	→	*The contents opened in the command line editor.*

See also: VISITB, EDIT, EDITB

VISITB

Type: Command

Description: For a specified variable, opens the contents in the most suitable editor for the object type. For example, if the specified variable holds an equation, the equation is opened in Equation Writer.

Access: [→] CAT VISITB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
A variable name	→	*The contents opened in the most suitable editor.*

See also: VISIT, EDIT, EDITB

VPOTENTIAL

Type: Command

Description: Find a vector potential function describing a field whose curl (or "rot") is the input. This command is the opposite of CURL. Given a vector *V* it attempts to return a function *U* such that curl *U* is equal to *V*; $\vec{\nabla} \times \vec{U} = \vec{V}$. For this to be possible, DIV(*V*) must be zero, otherwise the command reports a "Bad Argument Value" error. Step-by-step mode is available with this command.

Access: Catalog, [→] CAT

Input: Level 2/Argument 1: A vector *V* of expressions.
Level 1/Argument 2: A vector of the names of the variables.

Output: Level 1/Item 1: A vector *U* of the variables that is the potential from which *V* is obtained. An arbitrary constant can be added, the command does not do this.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –3 clear).
Radians mode must be set (flag –17 set).
Step-by-step mode can be set (flag –100 set).

Example: To see if this command is the opposite of CURL, use the output of the example in CURL as input to VPOTENTIAL. Find a vector in the spatial variables *x*, *y*, and *z* whose curl is: $(2yz)\boldsymbol{i} + (0)\boldsymbol{j} + (2xy - x^2)\boldsymbol{k}$

Command:
```
VPOTENTIAL([2*Y*Z, 0, 2*X*Y-X^2], [X,Y,Z])
EXPAND(ANS(1))
```

Result: `[0, -((X^3-3*Y*X^2)/3), Z*Y^2]`
This shows that the reversal is not unique – more than one vector can have the same curl.

∫ (Integrate)

Type: Function

Description: Integral Function: Integrates an *integrand* from *lower limit* to *upper limit* with respect to a specified variable of integration.

The algebraic syntax for ∫ parallels its stack syntax:

∫(*lower limit*, *upper limit*, *integrand*, *name*)

where *lower limit*, *upper limit*, and *integrand* can be real or complex numbers, unit objects, names, or algebraic expressions.

Evaluating ∫ in Symbolic Results mode (flag –3 *clear*) returns a symbolic result. Some functions that the calculator can integrate include the following:

- All built-in functions whose antiderivatives can be expressed in terms of other built-in functions — for example, SIN can be integrated since its antiderivative, COS, is a built-in function. The arguments for these functions must be linear.
- Sums, differences, and negations of built-in functions whose antiderivatives can be expressed in terms of other built-in functions — for example, 'SIN(X)–COS(X)'.
- Derivatives of all built-in functions — for example, 'INV(1+X^2)' can be integrated because it is the derivative of the built-in function ATAN.
- Polynomials whose base term is linear — for example, 'X^3+X^2–2*X+6' can be integrated since X is a linear term. '(X^2–6)^3+(X^2–6)^2' cannot be integrated since X^2–6 is not linear.
- Selected patterns composed of functions whose antiderivatives can be expressed in terms of other built-in functions — for example, '1/(COS(X)*SIN(X))' returns 'LN(TAN(X))'.

If the result of the integration is an expression with no integral sign in the result, the symbolic integration was successful. If, however, the result still contains an integral sign, try rearranging the expression and evaluating again, or estimate the answer using numerical integration.

Evaluating ∫ in Numerical Results mode (flag –3 *set*) returns a numerical approximation, and stores the error of integration in variable *IERR*. ∫ consults the number format setting to determine how accurately to compute the result.

Access: [→] ___∫ (___∫ is the right-shift of the [TAN] key).

Flags: Numerical Result (–3), Number Format (–45 to –50)

Input/Output:

L4/A1	L3/A2	L2/A3	L1/A4		L1/I1
lower limit	upper limit	integrand	'name'	→	'symb$_{integral}$'

L = Level; A = Argument; I = Item

Example: In Symbolic Results mode (flag –3 clear) this command sequence:

```
1 2 '10*X' 'X' ∫
```

returns 15.

In Numeric Results mode (flag –3 set) the above command sequence returns the numeric approximation 15.. In addition, the variable *IERR* is created, and contains the error of integration .00000000015.

See also: TAYLR, ∂, Σ

? (Undefined)

Type: Function

Description: The "undefined" symbol. Used to signify a numeric result that is not defined by the rules of arithmetic, such as the result of dividing zero by zero, or infinity by infinity. Mathematical operations on **?** return **?** as a result. Can be used in programs to check for an earlier undefined operation.

This use of **?** is unrelated to the use of **?** as a spare unit in the units system. The unit **?** can be used to create new units based on it, units that can not be expressed in terms of other base units. For

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	
0	→	x$_{key}$
–1	→	x$_{key}$

Example 1: This program:

```
« "Press [1] to add#Press any other key to subtract"
  1 DISP 0 WAIT IF 92.1 SAME THEN + ELSE - END »
```

displays a prompting message and halts program execution until a key is pressed. If the [1] key (location 92.1) is pressed, two numbers on the stack are added. If any other key is pressed, two numbers on the stack are subtracted.

Example 2: This program:

```
« { ADD { } { } { } { } SUB } MENU
  "Press [ADD] to add#Press [SUB] to subtract"
  1 DISP -1 WAIT IF 11.1 SAME THEN + ELSE - END »
```

builds a custom menu with labels ADD and SUB and a prompting message. Executing `-1 WAIT` displays the custom menu (note that it's not active) and suspends execution for keyboard input. If the ADD menu key (location 11.1) is pressed, two numbers on the stack are added. If any other key is pressed, two numbers on the stack are subtracted.

See also: KEY

WHILE

Type: Command Operation

Description: WHILE Indefinite Loop Structure Command: Starts the WHILE … REPEAT … END indefinite loop structure.

WHILE … REPEAT … END repeatedly evaluates a test and executes a loop clause if the test is true. Since the test clause occurs before the loop-clause, the loop clause is never executed if the test is initially false. The syntax is this:

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

The test clause is executed and must return a test result to the stack. REPEAT takes the value from the stack. If the value is not zero, execution continues with the loop clause; otherwise, execution resumes following END.

Access: [→] PRG BRANCH WHILE (PRG is the left-shift of the [EVAL] key).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
WHILE	→	
REPEAT	T/F →	
END	→	

See also: DO, END, REPEAT

WIREFRAME

Type: Command

Description: WIREFRAME Plot Type Command: Sets the plot type to WIREFRAME.

When the plot type is set to WIREFRAME, the DRAW command plots a perspective view of the graph of a scalar function of two variables. WIREFRAME requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
w	z	→	w^z
z	'symb'	→	'z^(symb)'
'symb'	z	→	'(symb)^z'
'symb$_1$'	'symb$_2$'	→	'symb$_1$^('symb$_2$)'
x_unit	y	→	x^y_unit^y
x_unit	'symb'	→	'(x_unit)^(symb)'

See also: EXP, ISOL, LN, XROOT

| (Where)

Type: Function

Description: Where Function: Substitutes values for names in an expression.

| is used primarily in algebraic objects, where its syntax is:

'*symb*$_{old}$ | (*name*$_1$ = *symb*$_1$, *name*$_2$ = *symb*$_2$ …)'

It enables algebraics to include variable-like substitution information about names. Symbolic functions that delay name evaluation (such as ∫ and ∂) can then extract substitution information from local variables and include that information in the expression, avoiding the problem that would occur if the local variables no longer existed when the local names were finally evaluated.

Access: ▶ ⌊ (⌊ is the right-shift of the TOOL key).

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'symb$_{old}$'	{ name$_1$, 'symb$_1$', name$_2$, 'symb$_2$' … }	→	'symb$_{new}$'
x	{ name$_1$, 'symb$_1$', name$_2$, 'symb$_2$' … }	→	x
(x,y)	{ name$_1$, 'symb$_1$', name$_2$, 'symb$_2$' … }	→	(x,y)

See also: APPLY, QUOTE

√ (Square Root)

Type: Function

Description: Square Root Analytic Function: Returns the (positive) square root of the argument.

For a complex number (x_1, y_1), the square root is this complex number:

$$(x_2, y_2) = \left(\sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}\right)$$

where r = ABS (x_1, y_1), and θ = ARG (x_1, y_1).
If (x_1, y_1) = (0,0), then the square root is (0, 0).
The inverse of SQ is a *relation*, not a function, since SQ sends more than one argument to the same result. The inverse relation for SQ is expressed by ISOL as this *general solution*:
's1*√Z'
The function √ is the inverse of a *part* of SQ, a part defined by restricting the domain of SQ such that:

1. each argument is sent to a distinct result, and
2. each possible result is achieved. The points in this restricted domain of SQ are called the *principal values* of the inverse relation. The √ function in its entirety is called the *principal branch* of the inverse relation, and the points sent by √ to the boundary of the restricted domain of SQ form the *branch cuts* of √.

Code	Description
5	A Deep Sleep wakeup (for example, ON, Alarm).
6	Not used
7	A 5-nibble word (CMOS test word) in RAM was corrupt. (This word is checked on every interrupt, but it is used only as an indicator of potentially corrupt RAM.)
8	Not used
9	The alarm list is corrupt.
A	System RPL jump to #0.
B	The card module was removed (or card bounce).
C	Hardware reset occurred (for example, an electrostatic discharge or user reset)
D	An expected System (RPL) error handler was not found in runstream.

The date and time stamp (*date time*) part of the log may be displayed as 00…0000 for one of three reasons:

- The system time was corrupt when the stamp was recorded.
- The date and time stamp itself is corrupt (bad checksum).
- Fewer than four warmstarts have occurred since the log was last cleared.

Access: ▶ CAT WSLOG

Flags: Date Format (–42)

Input/Output:

Level 1/Argument 1		Level 4/Item 1 ... Level 1/Item 4
	→	"log$_4$" ... "log$_1$"

ΣX

Type: Command

Description: Sum of *x*-Values Command: Sums the values in the independent-variable column of the current statistical matrix (reserved variable *ΣDAT*).

The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable *ΣPAR*. The default independent-variable column number is 1.

Access: ▶ CAT ΣX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x$_{sum}$

See also: NΣ, XCOL, ΣXY, ΣX2, ΣY, ΣY2

ΣX2

Type: Command

Description: Sum of Squares of *x*-Values Command: Sums the squares of the values in the independent-variable column of the current statistical matrix (reserved variable *ΣDAT*).

The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable *ΣPAR*. The default independent-variable column number is 1.

YVOL

Type: Command

Description: Y Volume Coordinates Command: Sets the depth of the view volume in the reserved variable *VPAR*.

The variables y_{near} and y_{far} are real numbers that set the *y*-coordinates for the view volume used in 3D plots. y_{near} must be less than y_{far}. These values are stored in the reserved variable *VPAR*.

Access: ↱ CAT YVOL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{near}	y_{far}	→	

See also: EYEPT, XVOL, XXRNG, YYRNG, ZVOL

YYRNG

Type: Command

Description: Y Range of an Input Plane (Domain) Command: Specifies the y range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

The variables $y_{y\ near}$ and $y_{y\ far}$ are real numbers that set the y-coordinates for the input plane. These values are stored in the reserved variable *VPAR*.

Access: ↱ CAT YYRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{near}	y_{far}	→	

See also: EYEPT, XVOL, XXRNG, YVOL, ZVOL

ZEROS

Type: Command

Description: Returns the zeros of a function of one variable, without multiplicity.

Access: SYMB SOLVE, Symbolic solve, ↰ S.SLV NXT

Input: Level 2/Argument 1: An expression.
Level 1/Argument 2: The variable to solve for.

Output: The solution, or a list of solutions, for the expression equated to 0.

Flags: Radians mode must be set (flag –17 set).
For a symbolic result, clear the CAS modes Numeric option (flag –3 clear).
The following flag settings affect the result:

- If Exact mode is set (flag –105 is clear), attempts to find exact solutions only. This may return a null list, even if approximate solutions exist.
- If Approximate mode is set (flag –105 set), finds numeric roots.
- If Complex mode is set (flag –103 set), searches for real and complex roots.

Example: Find the roots of the following equation in *x*, without specifying that *x*=2 is a root twice.
$x^3 - x^2 - 8x + 12 = 0$:

Command: `ZEROS(X^3-X^2-8*X+12)`

Results: `{-3, 2}`

After receiving an XOFF command (with *transmit pacing* in the reserved variable *IOPAR* set), XMIT stops transmitting and waits for an XON command. XMIT resumes transmitting if an XON is received before the time-out set by STIME elapses; otherwise, XMIT terminates, returns a 0, and stores "Timeout" in ERRM.

Access: ↱ CAT XMIT

Flags: I/O Device (–33), I/O Device for Wire (–78)

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
"string"	→		*1*
"string"	→	"substring$_{absent}$"	*0*

See also: BUFLEN, SBRK, SRECV, STIME

XNUM

Type: Command

Description: Converts an object or a list of objects to 12-digit decimal numeric format. Similar to →NUM except that →NUM does not work with lists, nor in programs in algebraic mode.

Access: Catalog, ↱ CAT

Input: An object or list of objects.

Output: The objects in numeric format.

Example: Find the 12-digit numeric values of π/2, 3*e*, and 4cos(2).

Command: `XNUM({π/2,3*e,4*COS(2)})`

Results: `{1.5707963268 8.15484548538 -1.66458734619}`

See also: I→R, →NUM

XOR

Type: Function

Description: Exclusive OR Function: Returns the logical exclusive OR of two arguments.

When the arguments are binary integers or strings, XOR does a bit-by-bit (base 2) logical comparison:

- Binary integer arguments are treated as sequences of bits with length equal to the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments, as shown in the following table:

bit_1	bit_2	bit_1 **XOR** bit_2
0	0	0
0	1	1
1	0	1
1	1	0

- String arguments are treated as sequences of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must be the same length.

When the arguments are real numbers or symbolics, XOR simply does a true/false test. The result is 1 (true) if either, but not both, arguments are nonzero; it is 0 (false) if both arguments are nonzero or zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic objects, then the result is an algebraic of the form $symb_1$ XOR $symb_2$. Execute →NUM (or set flag –3 before executing XOR) to produce a numeric result from the algebraic result.

Access: ↱ BASE NXT LOGIC XOR (BASE is the right-shift of the 3 key).

ΣY

Type: Command

Description: Sum of *y*-Values Command: Sums the values in the dependent variable column of the current statistical matrix (reserved variable *ΣDAT*).

The dependent variable column is specified by YCOL, and is stored as the second parameter in the reserved variable *ΣPAR*. The default dependent variable column number is 2.

Access: ▶ CAT ΣY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	Sum of Y

See also: NΣ, ΣX, XCOL, ΣXY, ΣX2, YCOL, ΣY2

ΣY2

Type: Command

Description: Sum of Squares of *y*-Values Command: Sums the squares of the values in the dependent-variable columns of the current statistical matrix (reserved variable *ΣDAT*). The dependent column is the column designated as YCOL

Access: ▶ CAT ΣY2

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	Sum of Y^2

See also: NΣ, ΣX, XCOL, ΣXY, ΣX2, YCOL

ΣY^2

Type: Command

Description: Sum of Squares of *y*-Values Command: Sums the squares of the values in the dependent-variable columns of the current statistical matrix. ΣY^2 is provided for compatibility with the HP 28. ΣY^2 is the same as ΣY2; see its listing for details.

Access:

YCOL

Type: Command

Description: Dependent Column Command: Specifies the dependent variable column of the current statistics matrix (reserved variable *ΣDAT*).

The dependent variable column number is stored as the second parameter in the reserved variable *ΣPAR*. The default dependent variable column number is 2.

YCOL will accept a noninteger real number and store it in *ΣPAR*, but subsequent commands that utilize the YCOL specification in *ΣPAR* will cause an error.

Access: ▶ CAT YCOL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{col}	→	

See also: BARPLOT, BESTFIT, COLΣ, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL

Example 1: Express .3658 in rational format, in Std mode:

Command: `XQ(.3658)`

Results: `1829/5000`

Example 2: Express .3658 in rational format, in Fix 4 mode:

Command: `XQ(.3658)`

Results: `√(19/142)`

Example 3: Express 1.04719755120 in rational format, in Eng 11 mode:

Command: `XQ(1.04719755120)`

Results: `1/3*π`

See also: →Q, →Qπ

XRECV

Type: Command

Description: XModem Receive Command: Prepares the calculator to receive an object via XModem. The received object is stored in the given variable name.

The transfer will start more quickly if you start the XModem sender *before* executing XRECV.

Invalid object names cause an error. If flag –36 is clear, object names that are already in use also cause an error.

If you are transferring data between two calculators, executing `{AAA BBB CCC} XRECV` receives *AAA*, *BBB*, and *CCC*. You also need to use a list on the sending end (`{AAA BBB CCC} XSEND`, for example).

Access: ▶ CAT XRECV

Flags: I/O Device (–33), RECV Overwrite (–36), I/O Device for Wire (–78)

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'name'	→	

See also: BAUD, RECV, RECN, SEND, XSEND

XRNG

Type: Command

Description: x-Axis Display Range Command: Specifies the *x*-axis display range.

The *x*-axis display range is stored in the reserved variable *PPAR* as x_{min} and x_{max} in the complex numbers (x_{min}, y_{min}) and (x_{max}, y_{max}). These complex numbers are the first two elements of *PPAR* and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of x_{min} and x_{max} are –6.5 and 6.5, respectively.

Access: ▶ CAT XRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{min}	x_{max}	→	

See also: AUTO, PDIM, PMAX, PMIN, YRNG

XROOT

Type: Analytic function

Description: *x*th Root of *y* Command: Computes the *x*th root of a real number.

XROOT is equivalent to $y^{1/x}$, but with greater accuracy.

If $y < 0$, *x* must be an integer.

Access: ▶ $\sqrt[x]{y}$ ($\sqrt[x]{y}$ is the right-shift of the $\sqrt{x}$ key).

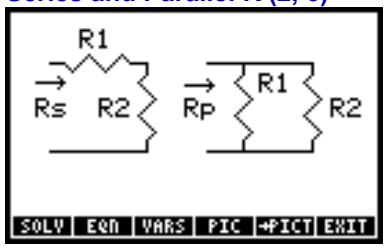
Flags: Numerical Results (–3)

Example:

Given: ρ=0.0035_Ω*cm, L=50_cm, A=1_cm^2.

Solution: R=0.175_Ω.

Series and Parallel R (2, 6)



Equation:

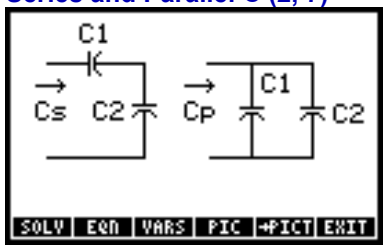
$$Rs = R1 + R2 \qquad \frac{1}{Rp} = \frac{1}{R1} + \frac{1}{R2}$$

Example:

Given: $R1$=2_Ω, $R2$=3_Ω.

Solution: Rs=5_Ω, Rp=1.2000_Ω.

Series and Parallel C (2, 7)



Equations:

$$\frac{1}{Cs} = \frac{1}{C1} + \frac{1}{C2} \qquad Cp = C1 + C2$$

Example:

Given: $C1$=2_μF, $C2$=3_μF.

Solution: Cs=1.2000_μF, Cp=5_μF.

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_1	obj_2	→	*0/1*
(x,0)	x	→	*0/1*
x	*(x,0)*	→	*0/1*
z	'symb'	→	'$z \neq$ symb'
'symb'	z	→	'symb ≠ z'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \neq symb_2$'

See also: SAME, TYPE, <, ≤, >,≥, ==, =

* (Multiply)

Type: Function

Description: Multiply Analytic Function: Returns the product of the arguments.

The product of a real number *a* and a complex number (*x*, *y*) is the complex number (*xa*, *ya*).

The product of two complex numbers (x_1, y_1) and (x_2, y_2) is the complex number ($x_1 x_2 - y_1 y_2$, $x_1 y_2 + x_2 y_1$).

The product of a real array and a complex array or number is a complex array. Each element *x* of the real array is treated as a complex element (*x*, 0).

Multiplying a matrix by an array returns a matrix product. The matrix must have the same number of columns as the array has rows (or elements, if it is a vector).

Although a vector is entered and displayed as a *row* of numbers, the calculator treats a vector as an $n \times 1$ matrix when multiplying matrices or computing matrix norms.

Multiplying a binary integer by a real number returns a binary integer that is the product of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the multiplication.)

The product of two binary integers is truncated to the current binary integer wordsize.

When multiplying two unit objects, the scalar parts and the unit parts are multiplied separately.

Access: ⊠

Flags: Numerical Results (–3), Binary Integer Wordsize (–5 through –10)

Variable	Description
Qp,Qs	Parallel and series quality factors
r	Charge distance
R,R1,R2	Resistance
ri,ro	Inside and outside radii
Rp,Rs	Parallel and series resistances
t	Time
ti,tf	Initial and final times
V	Voltage, or Total voltage (Voltage Divider)
V1	Voltage across R1
Vi,Vf	Initial and final voltages
Vmax	Maximum voltage
XC	Reactance of capacitor
XL	Reactance of inductor

Reference: 3.

Coulomb's Law (2, 1)

This equation describes the electrostatic force between two charged particles.

Equation:

$$F = \frac{1}{4 \cdot \pi \cdot \epsilon 0 \cdot \epsilon r} \cdot \left(\frac{q1 \cdot q2}{r^2}\right)$$

Example:

Given: *q1*=1.6E-19_C, *q2*=1.6E-19_C, *r*=4.00E-13_cm, ϵr=1.00.

Solution: *F*=14.3801_N.

Ohm's Law and Power (2, 2)

Equations:

$$V = I \cdot R \qquad P = V \cdot I \qquad P = I^2 \cdot R \qquad P = \frac{V^2}{R}$$

Example:

Given: *V*=24_V, *I*=16_A.

Solution: *R*=1.5_Ω, *P*=384_W.

and %T treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Access: (+)

Flags: Numerical Results (–3), Binary Integer Wordsize (–5 through –10)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
z_1	z_2	→	$z_1 + z_2$
⌈ array ⌋$_1$	⌈ array ⌋$_2$	→	⌈ array ⌋$_3$
z	'symb'	→	'z +symb'
'symb'	z	→	'symb +z'
'symb$_1$'	'symb$_2$'	→	'symb$_1$ + symb$_2$'
{ list$_1$ }	{ list$_2$	→	{ list$_1$ list$_2$ }
obj$_4$	{ obj$_1$ … obj$_n$ }	→	{ obj$_4$ obj$_1$ … obj$_n$ }
{ obj$_1$ … obj$_n$ }	obj$_4$	→	{obj$_1$ … obj$_n$ obj$_4$ }
"string$_1$"	"string$_2$"	→	"string$_1$ string$_2$"
obj	"string"	→	"obj string"
"string"	obj	→	"string obj"
#n$_1$	n$_2$	→	#n$_3$
n$_1$	#n$_2$	→	#n$_3$
#n$_1$	#n$_2$	→	#n$_3$
x$_1$_unit$_1$	y_unit$_2$	→	(x$_2$ + y)_unit$_2$
'symb'	x_unit	→	'symb + x_unit'
x_unit	'symb'	→	'x_unit + symb'
grob$_1$	grob$_2$	→	grob$_3$

Example 1: `{ 1 2 3 } { A B C } +` returns `{ 1 2 3 A B C }`.

Example 2: `5_ft 9_in +` returns `69_in`.

Example 3: `[[ 0 1 ][ 1 3 ]] [[ 2 1 ][ 0 1 ]] +` returns `[[ 2 2 ][ 1 4 ]]`.

Example 4: `'FIRST' 'SECOND' +` returns `'FIRST+SECOND'`.

See also: –, *, /, =, ADD

– (Subtract)

Type: Function

Description: Subtract Analytic Function: Returns the difference of the arguments.

The difference of a real number *a* and a complex number (*x*, *y*) is (*x*–*a*, *y*) or (*a*–*x*, –*y*). The difference of two complex numbers (x_1, y_1) and (x_2, y_2) is ($x_1 - x_2$, $y_1 - y_2$).

The difference of a real array and a complex array is a complex array, where each element *x* of the real array is treated as a complex element (*x*, 0). The two array arguments must have the same dimensions.

The difference of a binary integer and a real number is a binary integer that is the sum of the first argument and the two's complement of the second argument. (The real number is converted to a binary integer before the subtraction.)

Equation:

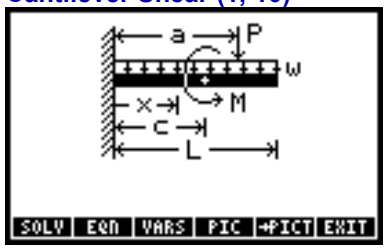
$$Mx = P \cdot (x - a) + M - \frac{W}{2} \cdot (L^2 - 2 \cdot L \cdot x + x^2)$$

Example:

Given: L=10_ft, P=500lbf, M=800_ft*lbf, a=3_ft, c=6_ft, w=100_lbf/ft, x=8_ft.

Solution: Mx= -200_ft*lbf

Cantilever Shear (1, 10)



Equation:

$$V = P + w \cdot (L - x)$$

Example:

Given: L=10_ft, P=500lbf, a=3_ft, x=8_ft, w=100_lbf/ft.

Solution: V=200_lbf

$$\left(\frac{ax}{x^2+y^2}, \frac{ay}{x^2+y^2}\right)$$

A complex number (x, y) divided by a real number a returns the complex number $(x/a, y/a)$.

A complex number (x_1, y_1) divided by another complex number (x_2, y_2) returns this complex quotient:

$$\left(\frac{x_1x_2+y_1y_2}{x_2^2+y_2^2}, \frac{y_1x_2-x_1y_2}{x_2^2+y_2^2}\right)$$

An array **B** divided by a matrix **A** solves the system of equations **AX**=**B** for **X**; that is, **X** = **A**$^{-1}$ **B**. This operation uses 15-digit internal precision, providing a more precise result than the calculation INV(A)*B. The matrix must be square, and must have the same number of columns as the array has rows (or elements, if the array is a vector).

A binary integer divided by a real or binary number returns a binary integer that is the integral part of the quotient. (The real number is converted to a binary integer before the division.) A divisor of zero returns # 0.

When dividing two unit objects, the scalar parts and the unit parts are divided separately.

Access: ÷

Flags: Numerical Results (–3)

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
z_1	z_2	→	z_1 / z_2
[array]	[[matrix]]	→	[[matrix^{-1} ×array]]
z	'symb'	→	'z /symb'
'symb'	z	→	'symb /z'
'symb$_1$'	'symb$_2$'	→	'symb$_1$ / symb$_2$'
#n$_1$	n$_2$	→	#n$_3$
n$_1$	#n$_2$	→	#n$_3$
#n$_1$	#n$_2$	→	#n$_3$
x_unit$_1$	y_unit$_2$	→	(x / y)_unit$_1$/unit$_2$
x	y_unit	→	(x / y)_1/unit
x_unit	y	→	(x / y)_unit
'symb'	x_unit	→	'symb / x_unit'
x_unit	'symb'	→	'x_unit / symb'

See also: +, –, *, =

= (Equal)

Type: Function

Description: Equals Analytic Function: Returns an equation formed from the two arguments.

The equals sign equates two expressions such that the difference between them is zero.

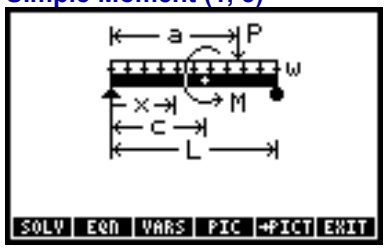
In Symbolic Results mode, the result is an algebraic equation. In Numerical Results mode, the result is the difference of the two arguments because = acts equivalent to –. This allows expressions and equations to be used interchangeably as arguments for symbolic and numerical rootfinders.

Example:

Given: L=20_ft, E=29000000_psi, I=40_in^4, a=10_ft, P=674.427_lbf, c=17_ft, M=3687.81_ft*lbf, w=102.783_lbf/ft, x=9_ft.

Solution: Θ= -0.0876_°.

Simple Moment (1, 5)



Equation:

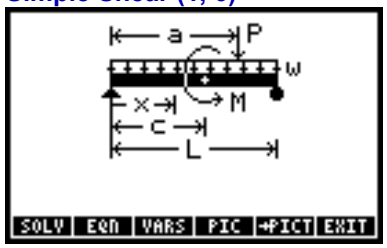
$$Mx = \frac{P \cdot (L - a) \cdot x}{L} + \frac{M \cdot x}{L} + \frac{w \cdot x}{2} \cdot (L - x)$$

Example:

Given: L=20_ft, a=10_ft, P=674.427_lbf, c=17_ft, M=3687.81_ft*lbf, w=102.783_lbf / ft, x=9_ft.

Solution: Mx=9782.1945_ft*lbf

Simple Shear (1, 6)



Equation:

$$V = \frac{P \cdot (L - a)}{L} + \frac{M}{L} + \frac{w}{2} \cdot (L - 2 \cdot x)$$

Example:

Given: L=20_ft, a=10_ft, P=674.427_lbf, M=3687.81_ft*lbf, w=102.783_lbf/ft, x=9_ft.

Solution: V=624.387_lbf.

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_1	obj_2	→	*0/1*
(x,0)	x	→	*0/1*
x	*(x,0)*	→	*0/1*
z	'symb'	→	'z == symb'
'symb'	z	→	'symb == z'
$'symb_1'$	$'symb_2'$	→	$'symb_1 == symb_2'$

See also: SAME, TYPE, <, ≤, >, ≥, ≠

▶ (Store)

Type: Command

Description: Store Command: Stores an object into a specified variable. To create a backup object, store the *obj* into the desired backup location (identified as $:n_{port}:name_{backup}$). ▶ will not overwrite an existing backup object. To replace an element of an array or list, use STO. Also use STO to store a graphic object into PICT or a library or backup object into a port.

Access: STO▶

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'name'	→	obj
obj	$:n_{port}:name_{backup}$	→	obj

See also: DEFINE, RCL, →, STO

→ (Create Local)

Type: Command

Description: Create Local Variables Command: Creates local variables.

Local variable structures specify one or more local variables and a defining procedure.

A local variable structure consists of the → command, followed by one or more names, followed by a defining procedure — either a program or an algebraic. The → command stores objects into local variables with the specified names. The resultant *local variables* exist only while the defining procedure is being executed. The syntax of a local variable structure is one of the following:

- → $name_1$ $name_2$ … $name_n$ « *program* »
- → $name_1$ $name_2$ … $name_n$ '*algebraic expression*'

Access: ↱ → (→ is the right-shift of the 0 key).

Input/Output:

Levelⁿ/Argument1 … Level1/Argumentn		Level 1/Item 1
obj_1 … obj_n	→	

Example 1: This program:

« → x y « x y * x y - + »

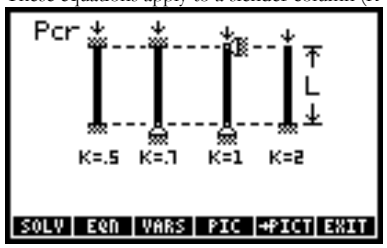
takes an object from level 2 and stores it in local variable *x*, takes an object from level 1 and stores it in local variable *y*, and executes calculations with *x* and *y* in the defining procedure (in this case a program). When the defining procedure ends, local variables *x* and *y* disappear.

Example 2: A user-defined function is a variable containing a program that consists solely of a local variable structure.

For example, the variable *A*, containing this program:

Elastic Buckling (1, 1)

These equations apply to a slender column ($K \cdot L / r > 100$) with length factor K.



Equations:

$$Pcr = \frac{\pi^2 \cdot E \cdot A}{\left(\frac{K \cdot L}{r}\right)^2} \qquad Pcr = \frac{\pi^2 \cdot E \cdot I}{(K \cdot L)^2} \qquad \sigma cr = \frac{Pcr}{A} \qquad r = \sqrt{\frac{I}{A}}$$

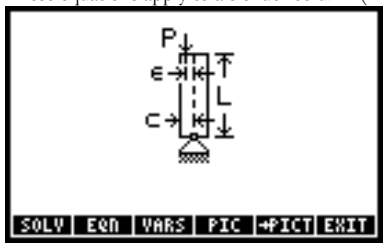
Example:

Given: L=7.3152_m, r=4.1148_cm, E=199947961.502_kPa, A=53.0967_cm^2, K=0.7, I=8990598.7930_mm^4.

Solution: Pcr=676.6019_kN, σcr=127428.2444_kPa.

Eccentric Columns (1, 2)

These equations apply to a slender column ($K \cdot L / r > 100$) with length factor K.



Equations:

$$\sigma max = \frac{P}{A} \cdot \left(1 + \frac{e \cdot c}{r^2} \cdot \left(\frac{1}{\cos\left(\frac{K \cdot L}{2 \cdot r} \cdot \sqrt{\frac{P}{E \cdot A}}\right)}\right)\right) \qquad r = \sqrt{\frac{I}{A}}$$

Example:

Given: L=6.6542_m, A=187.9351_cm^2, r=8.4836_cm, E=206842718.795_kPa, I=135259652.16_mm^4, K=1, P=1908.2571_kN, c=15.24_cm, e=1.1806_cm.

Solution: σmax=140853.0970_kPa.

Computer Algebra System

CAS Settings

Selecting CAS Settings

CAS settings are selected using the CAS MODES input form, described in Chapter 1 of the User's Manual. Selecting a mode is equivalent to setting or clearing one of the system flags, the flag numbers are given in the "Flags" part of the operation descriptions.

Pressing the NXT key in the CAS MODES input form displays a menu that allows the user to calculate settings. For example, if the Modulo field is selected in the CAS MODES form, and NXT is pressed, the following menu keys are available.

TYPE lists the types of object that can be chosen for this setting. For the modulo, this can be a real number or an integer. For checked mode settings, this can only be a real number; if it is zero the mode is unchecked, if it is anything else the mode is checked.

EVAL lets the user calculate a value for the setting, for example a new value for the modulo setting can be calculated. The HOME menu key allows the user to switch the heading lines between the "CAS MODES" heading and the normal heading lines, so that the user can see what the current settings are while carrying out a calculation.

RESET allows the setting to be reset to its default value, or all CAS settings to be reset to their default values.

See Appendix C in the User's Guide for further details of the CAS settings, and for other information about the CAS. Information on the Help system of the CAS is provided in Appendix C and also in Appendix H of the User's Guide.

The CAS directory, CASDIR

CAS settings are stored as flag settings, and as variables in the CASDIR directory, which is automatically created as a subdirectory of the HOME directory. Variables in this directory include:

VX: A name or list of names of the current CAS variable or variables. Default value is X

MODULO: The current modulus used for CAS modulo operations. Default value is 13, but is reset to 3 by the RESET key in the CAS menu.

PERIOD: The period for CAS periodic operations, 2π by default.

EPS: The value chosen such that coefficients in a polynomial smaller than this value are replaced with 0 by the EPSX0 command. 1E-10 by default.

REALASSUME: A list of the names of variables that some CAS operations treat as real numbers when complex mode is set. If additional assumptions are made on any variables, these are included here. By default the list is {X, Y, t, S1, S2}.

PRIMIT: Temporary storage of anti-derivative expressions used during CAS operations.

MATRIX: Temporary storage of a matrix used during CAS operations.

CASINFO: Temporary storage of graphic display during step-by-step operations.

See Appendix D for a complete list of CASDIR variables.

Points to note when choosing settings

The CAS is a powerful tool, and part of that power lies in the many modes and settings available. This means that if a setting is wrong then the CAS can give unexpected results or error messages. The following points should be observed. If an unexpected error occurs, or an unexpected message is seen, check this list.

Subject, var (subj, title)	Pic	EQ	Pg
COLUMNS AND BEAMS, 22 ********(1)*****			
Elastic Buckling (1,1)	Y	4	5-4
Eccentric Columns (1,2)	Y	2	5-4
Simple Deflection (1,3)	Y	1	5-5
Simple Slope (1,4)	Y	1	5-5
Simple Moment (1,5)	Y	1	5-6
Simple Shear (1,6)	Y	1	5-6
Cantilever Deflection (1,7)	Y	1	5-7
Cantilever Slope (1,8)	Y	1	5-7
Cantilever Moment (1,9)	Y	1	5-7
Cantilever Shear (1,10)	Y	1	5-8
ELECTRICITY, 47 *************(2)*****			
Coulomb's Law (2,1)	N	1	5-10
Ohm's Law and Power (2,2)	N	4	5-10
Voltage Divider (2,3)	Y	1	5-11
Current Divider (2,4)	Y	1	5-11
Wire Resistance (2,5)	Y	2	5-11
Series and Parallel R (2,6)	Y	2	5-12
Series and Parallel C (2,7)	Y	2	5-12
Series and Parallel L (2,8)	Y	2	5-13
Capacitive Energy (2,9)	N	1	5-13
Inductive Energy (2,10)	N	1	5-13
RLC Current Delay (2,11)	Y	5	5-14
DC Capacitor Current (2,12)	N	3	5-14
Capacitor Charge (2,13)	N	1	5-14
DC Inductor Voltage (2,14)	N	3	5-15
RC Transient (2,15)	Y	1	5-15
RL Transient (2,16)	N	1	5-15
Resonant Frequency (2,17)	N	4	5-16
Plate Capacitor (2,18)	Y	1	5-16
Cylindrical Capacitor (2,19)	Y	1	5-16
Solenoid Inductance (2,20)	Y	1	5-17
Toroid Inductance (2,21)	Y	1	5-17
Sinusoidal Voltage (2,22)	N	2	5-18
Sinusoidal Current (2,23)	N	2	5-18
FLUIDS, 24 ******************(3)*****			
Pressure at Depth (3,1)	Y	1	5-19
Bernoulli Equation (3,2)	Y	10	5-19
Flow with Losses (3,3)	Y	10	5-20
Flow in Full Pipes (3,4)	Y	8	5-21
FORCES AND ENERGY, 32 ********(4)*****			
Linear Mechanics (4,1)	N	8	5-22
Angular Mechanics (4,2)	N	12	5-23
Centripetal Force (4,3)	N	4	5-23
Hooke's Law (4,4)	Y	2	5-23
1D Elastic Collisions (4,5)	Y	2	5-24
Drag Force (4,6)	N	1	5-24
Law of Gravitation (4,7)	N	1	5-24
Mass-Energy Relation (4,8)	N	1	5-24
GASES, 26 ********************(5)*****			
Ideal Gas Law (5,1)	N	2	5-25
Ideal Gas State Change (5,2)	N	1	5-26
Isothermal Expansion (5,3)	N	2	5-26
Polytropic Processes (5,4)	N	2	5-26
Isentropic Flow (5,5)	Y	4	5-26
Real Gas Law (5,6)	N	2	5-27
Real Gas State Change (5,7)	N	1	5-27
Kinetic Theory (5,8)	N	4	5-28

Subject, var (subj, title)	Pic	EQ	Pg
HEAT TRANSFER, 23 ************(6)*****			
Heat Capacity (6,1)	N	2	5-29
Thermal Expansion (6,2)	Y	2	5-29
Conduction (6,3)	Y	2	5-29
Convection (6,4)	Y	2	5-30
Conduction+Convection (6,5)	Y	4	5-30
Black Body Radiation (6,6)	Y	5	5-31
MAGNETISM, 11 ****************(7)*****			
Straight Wire (7,1)	Y	1	5-32
Force between Wires (7,2)	Y	1	5-32
Magnetic (B) Field in Solenoid (7,3)	Y	1	5-33
Magnetic (B) Field in Toroid (7,4)	Y	1	5-33
MOTION, 30 ******************(8)*****			
Linear Motion (8,1)	N	4	5-35
Object in Free Fall (8,2)	N	4	5-35
Projectile Motion (8,3)	Y	5	5-35
Angular Motion (8,4)	N	4	5-36
Circular Motion (8,5)	N	3	5-36
Terminal Velocity (8,6)	N	1	5-36
Escape Velocity (8,7)	N	1	5-36
OPTICS, 10 ******************(9)*****			
Law of Refraction (9,1)	Y	1	5-37
Critical Angle (9,2)	Y	1	5-38
Brewster's Law (9,3)	Y	2	5-38
Spherical Reflection (9,4)	Y	3	5-39
Spherical Refraction (9,5)	Y	1	5-39
Thin Lens (9,6)	Y	3	5-39
OSCILLATIONS, 17 ***********(10)*****			
Mass-Spring System (10,1)	Y	3	5-41
Simple Pendulum (10,2)	Y	3	5-41
Conical Pendulum (10,3)	Y	4	5-42
Torsional Pendulum (10,4)	Y	3	5-42
Simple Harmonic (10,5)	N	4	5-42
PLANE GEOMETRY, 21 **********(11)*****			
Circle (11,1)	Y	5	5-44
Ellipse (11,2)	Y	5	5-44
Rectangle (11,3)	Y	5	5-45
Regular Polygon (11,4)	Y	6	5-45
Circular Ring (11,5)	Y	4	5-46
Triangle (11,6)	Y	5	5-46
SOLID GEOMETRY, 12 **********(12)*****			
Cone (12,1)	Y	5	5-47
Cylinder (12,2)	Y	5	5-48
Parallelpiped (12,3)	Y	4	5-48
Sphere (12,4)	Y	4	5-49
SOLID STATE DEVICES, 60 *****(13)*****			
PN Step Junctions (13,1)	Y	8	5-52
NMOS Transistors (13,2)	Y	10	5-53
Bipolar Transistors (13,3)	Y	8	5-54
JFETs (13,4)	Y	7	5-54
STRESS ANALYSIS, 28 *********(14)*****			
Normal Stress (14,1)	Y	3	5-57
Shear Stress (14,2)	Y	3	5-57
Stress on an Element (14,3)	Y	3	5-57
Mohr's Circle (14,4)	Y	7	5-58
WAVES, 16 ******************(15)***			
Transverse Waves (15,1)	Y	4	5-59
Longitudinal Waves (15,2)	N	4	5-59
Sound Waves (15,3)	N	4	5-60

Using the CAS

Examples and Help

In addition to the examples in this Command Reference, the built-in CAS help provides examples of CAS operations.

- If an operation is selected from the operations catalog, ▶ CAT, and if help is available, then pressing the ⓗ key shows help information. Pressing the EX menu key copies the operation to the command line, ready for use.
- If an operation is selected from the CASCMD list, TOOL NXT CASCMD, the same help information is available, but instead of EX, there is an ECHO key to copy the name and example to the command line. Evaluating the example and comparing it with the result shown in the help text is a quick way to check if the CAS settings are correct.

Compatibility with Other Calculators

Some CAS operations replace similar operations that were available on older HP calculators. The older operation names have been kept on the HP 50g, HP 49g+, and HP 48gII so that programs written for the older calculators will work on the new models without being rewritten. This means that some commands and functions have more than one name; these are indicated as such within the Command Reference in Chapter 3.

The older models whose programs can be run on the HP 50g, HP 49g+, and HP 48gII are the HP 28C and HP 28S, the HP 48S and the HP 48SX, and the HP 48G, HP 48GX and HP 48G+. These models only had the RPL programming language, so programs written for them should be used in RPN mode. The HP 49G is a more recent model which does have the CAS, and Algebraic mode, so programs written for it in either RPL or in Algebraic mode can be used on the HP 50g, HP 49g+, and HP 48gII. The CAS of the HP 50g, HP 49g+, and HP 48gII is also very similar to the CAS of the HP 40G and HP 40gs models, so programs and books written for them may be helpful.

Extending the CAS

Users can extend the CAS by writing their own functions or commands. Functions can be written as UDFs (User Defined Functions); see the description of DEFINE in Chapter 3 of the calculator User's Guide and the descriptions of DEFINE and DEF in the Command Reference in Chapter 3 of this reference. The pattern matching commands ↑MATCH and ↓MATCH allow the user to write programs to edit algebraic expressions. Here is an example of an RPL program using ↑MATCH to replace the square root of a square of a symbol with the symbol itself. The wildcard &A means that any symbol or expression squared can be replaced. The conditional expression &A≥0 means that the replacement is only carried out if the square root is not of a negative value.

```
« { '√(&A^2)' &A '&A≥0' } ↑MATCH »
```

Dealing with unexpected CAS results or messages

If a CAS operation gives an unexpected result or message, check the list of points given in the section on CAS settings. Some problems can be caused by unexpected settings, so it can be helpful to reset all CAS settings to their default values, with the CASCFG command, or with the RESET key in the CAS settings menu.

CAS menu commands, ↱ CAT

These commands display menus or lists of CAS operations.

CAS utility operations

Arithmetic Modulo commands, ↰ ARITH MODULO

Arithmetic Permutation commands, ↰ ARITH PERMUTATION

Other Arithmetic commands, ↰ ARITH

Convert commands, ↰ CONVERT

Unit conversion tools, ↰ CONVERT UNITS TOOLS

Base conversion tools, ↰ CONVERT BASE

All operations in the BASE submenus are described in Chapter 3.

Trigonometric conversions, ↰ CONVERT TRIG CONV

Rewrite expression, ↰ CONVERT REWRITE

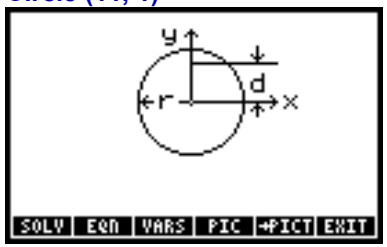
Exp and Lin commands, ↰ EXP&LN

Matrix-related commands

Create, ↰ MATRICES CREATE

Operations, ↰ MATRICES OPERATIONS

Circle (11, 1)



Equations:

$$A = \pi \cdot r^2 \qquad C = 2 \cdot \pi \cdot r \qquad I = \frac{\pi \cdot r^4}{4}$$

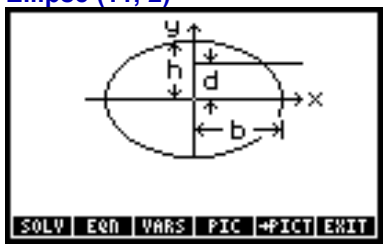
$$J = \frac{\pi \cdot r^4}{2} \qquad Id = I + A \cdot d^2$$

Example:

Given: *r*=5_cm, *d*=1.5_cm.

Solution: *C*=31.4159_cm, *A*=78.5398_cm^2, *I*=4908738.5_mm^4, *J*=9817477.0_mm^4, *Id*=6675884.4_mm^4.

Ellipse (11, 2)



Equations:

$$A = \pi \cdot b \cdot h \qquad C = 2 \cdot \pi \cdot \sqrt{\frac{b^2 + h^2}{2}} \qquad I = \frac{\pi \cdot b \cdot h^3}{4}$$

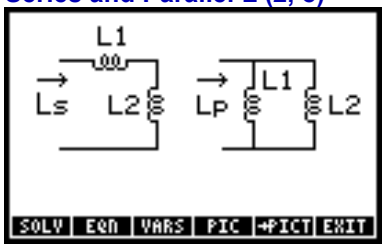
$$J = \frac{\pi \cdot b \cdot h}{4} \cdot (b^2 + h^2) \qquad Id = I + A \cdot d^2$$

Equations: Example:

Given: *b*=17.85_µm, *b*=78.9725_µin, *d*=.00000012_ft.

Solution: *A*=1.1249E–6_cm^2, C=7.9805E–3_cm, *I*=1.1315E–10_mm^4, *J*=9.0733E–9_mm^4, *Id*=1.1330E–10_mm^4.

Series and Parallel L (2, 8)



Equations:

$$Ls = L1 + L2 \qquad \frac{1}{Lp} = \frac{1}{L1} + \frac{1}{L2}$$

Example:

Given: *L1*=17_mH, *L2*=16.5_mH,

Solution: *Ls*=33.5000_mH, *Lp*=8.3731_mH.

Capacitive Energy (2, 9)

Equation:

$$E = \frac{C \cdot V^2}{2}$$

Example:

Given: *E*=0.025_J, *C*=20_µF.

Solution: *V*=50_V.

Inductive Energy (2, 10)

Equation:

$$E = \frac{L \cdot I^2}{2}$$

Example:

Given: *E*= 4_J, *L*=15_mH.

Solution: *I*=23.0940_A.

Conical Pendulum (10, 3)

Equations:

$$\omega = \sqrt{\frac{g}{h}} \qquad h = L \cdot \cos(\theta) \qquad T = \frac{2 \cdot \pi}{\omega}$$

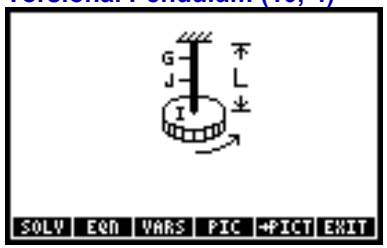
$$\omega = 2 \cdot \pi \cdot f$$

Example:

Given: L=25_cm, h=20_cm.

Solution: θ=36.899_°, T= 0.8973_s, ω=7.0024 r/s, f=1.1145_Hz.

Torsional Pendulum (10, 4)



Equations:

$$w = \sqrt{\frac{G \cdot J}{L \cdot I}} \qquad T = \frac{2 \cdot \pi}{\omega} \qquad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: G=1000_kPa, J=17_mm^4, L=26_cm, I=50_kg*m^2.

Solution: ω=1.1435E°–3_r/s, f=1.8200E–4_Hz, T=5494.4862_s.

Simple Harmonic (10, 5)

Equations:

$$x = xm \cdot \cos(\omega \cdot t + \phi) \qquad v = -\omega \cdot xm \cdot \sin(\omega \cdot t + \phi)$$

$$a = -\omega^2 \cdot xm \cdot \cos(\omega \cdot t + \phi) \qquad \omega = 2 \cdot \pi \cdot f$$

DC Inductor Voltage (2, 14)

These equations approximate the dc voltage induced in an inductor by a change in current in a certain time interval.

Equations:

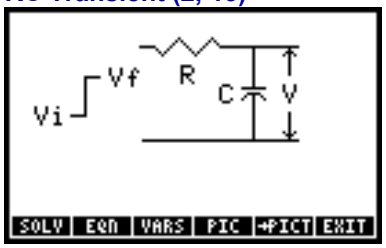
$$V = -V \cdot \left(\frac{\Delta I}{\Delta t}\right) \qquad \Delta V = -If - Ii \qquad \Delta t = tf - ti$$

Example:

Given: L=100_mH, V=52_V, ⊿t=32_μs, Ii=23_A, ti=0_s.

Solution: ⊿I= -0.0166_A, If=22.9834_A, tf=32_μs.

RC Transient (2, 15)



Equation:

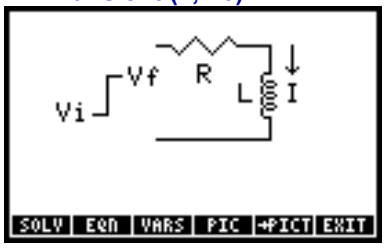
$$V = Vf - (Vf - Vi) \cdot e^{\frac{-t}{R \cdot C}}$$

Example:

Given: Vi=0_V, C=50_μF, Vf=10_V, R=100_ω, t=2_ms.

Solution: V=3.2968_V.

RL Transient (2, 16)

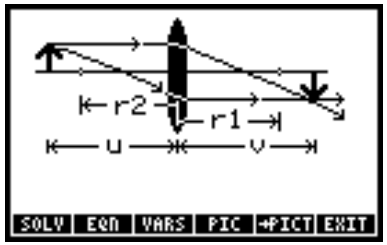


Equation:

$$I = \frac{1}{R} \cdot \left(Vf - (Vf - Vi) \cdot e^{\frac{-t \cdot R}{L}}\right)$$

Example:

Given: Vi=0_V, Vf=5_V, R=50_ω, L=50_mH, t=75_μs.



Equations:

$$\frac{1}{u}+\frac{1}{v}=\frac{1}{f} \qquad \frac{1}{f}=(n-1)\cdot\left(\frac{1}{r1}-\frac{1}{r2}\right) \qquad m=\frac{-v}{u}$$

Example:

Given: $r1$=5_cm, $r2$=20_cm, n=1.5, u=50_cm.

Solution: f=13.3333_cm, v=18.1818_cm, m= -0.3636.

Oscillations (10)

Variable	Description
ω	Angular frequency
ϕ	Phase angle
θ	Cone angle
a	Acceleration at t
f	Frequency
G	Shear modulus of elasticity
h	Cone height
I	Moment of inertia
J	Polar moment of inertia
k	Spring constant
L	Length of pendulum
m	Mass
t	Time
T	Period
v	Velocity at t
x	Displacement at t
xm	Displace amplitude

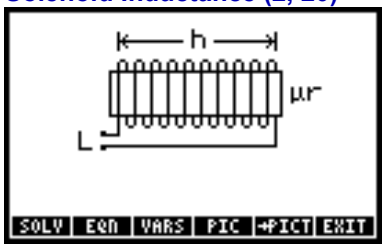
Reference: 3.

Example:

Given: ϵr=1, ro=1_cm, ri=.999_cm, L=10_cm.

Solution: C=0.0056_μF.

Solenoid Inductance (2, 20)



Equation:

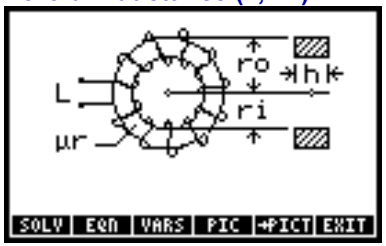
$$L=\mu 0\cdot\mu r\cdot n^2\cdot A\cdot h$$

Example:

Given: μr=2.5, n=40_1/cm, A= .2_cm^2, h=3_cm.

Solution: L=0.0302_mH.

Toroid Inductance (2, 21)



Equation:

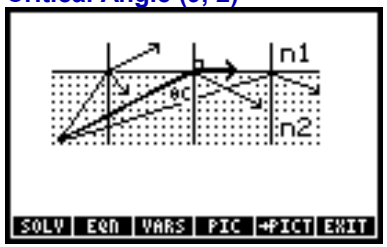
$$L=\frac{\mu 0\cdot\mu r\cdot N^2\cdot h}{2\cdot\pi}\cdot LN\left(\frac{ro}{ri}\right)$$

Example:

Given: μr=1, N=5000, h=2_cm, ri= .2_cm, ro=4_cm.

Solution: L=69.3147_mH.

Critical Angle (9, 2)



Equation:

$$\mathrm{SIN}(\theta c) = \frac{n1}{n2}$$

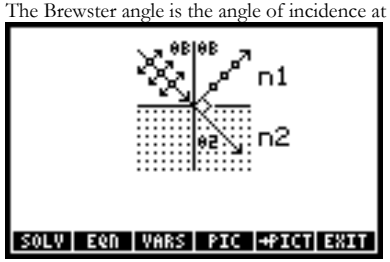
Example:

Given: *n1*=1, *n2*=1.5.

Solution: *θc*=41.8103_°.

Brewster's Law (9, 3)

The Brewster angle is the angle of incidence at which the reflected wave is completely polarized.



Equations:

$$\mathrm{TAN}(\theta B) = \frac{n2}{n1} \qquad \theta B + \theta 2 = 90$$

Example:

Given: *n1*=1, *n2*=1.5.

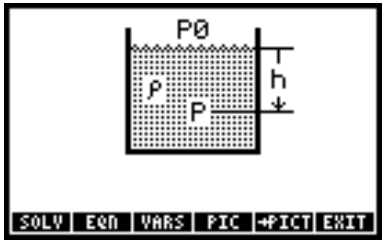
Solution: *θB*=56.3099_°, *θ2*=33.6901_°.

Variable	Description
Re	Reynolds number
v1, v2	Initial and final velocities
vavg	Average velocity
W	Power input
y1, y2	Initial and final heights

References: 3,6,9.

Pressure at Depth (3, 1)

This equation describes hydrostatic pressure for an incompressible fluid. Depth *h* is positive downwards from the reference.



Equation:

$$P = P0 + \rho \cdot g \cdot h$$

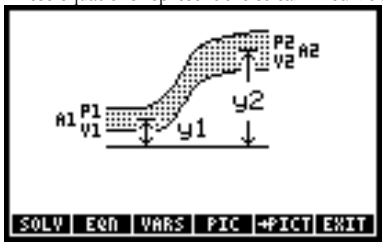
Example:

Given: *h*=100_m, *ρ*=1025.1817_kg/m^3, *P0*=1_atm.

Solution: *P*=1106.6848_kPa.

Bernoulli Equation (3, 2)

These equations represent the streamlined flow of an incompressible fluid.



Angular Motion (8, 4)

Equations:

$$\theta = \theta 0 + \omega 0 \cdot t + \frac{1}{2} \cdot \alpha \cdot t^2 \qquad \theta = \theta 0 + \omega \cdot t + \frac{1}{2} \cdot \alpha \cdot t^2$$

$$\theta = \theta 0 + \frac{1}{2} \cdot (\omega 0 + \omega) \cdot t \qquad \omega = \omega 0 + \alpha \cdot t$$

Example:

Given: *Θ0*=0_°, *ω0*=0_r/min, *α*=1.5_r/min^2, *t*=30_s.

Solution: *Θ*=10.7430_°, *ω*= 0.7500_r/min.

Circular Motion (8, 5)

Equations:

$$\omega = \frac{v}{r} \qquad ar = \frac{v^2}{r} \qquad \omega = 2 \cdot \pi \cdot N$$

Example:

Given: *r*=25_in, *v*=2500_ft/s

Solution: ω=72000_r/min, *ar*=3000000_ft/s^2, *N*=11459.1559_rpm.

Terminal Velocity (8, 6)

Equation:

$$v = \sqrt{\frac{2 \cdot m \cdot g}{Cd \cdot \rho \cdot A}}$$

Example:

Given: *Cd*=0.15, *ρ*=0.025lb/ft^3, *A*=100000_in^2, *m*=1250_lb.

Solution: *v*=1757.4709_ft/s.

Escape Velocity (8, 7)

Equation:

$$v = \sqrt{\frac{2 \cdot G \cdot M}{R}}$$

Example:

Given: *M*=1.5E23_lb, *R*=5000_mi.

Solution: *v*=3485.1106_ft/s.

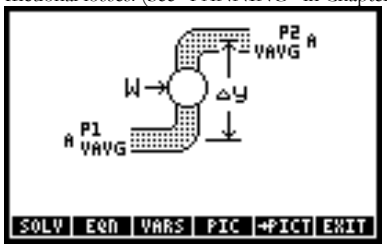
Example:

Given: *P2*=30_psi, *P1*=65_psi, *y2*=100_ft, *y1*=0_ft, *ρ*=64_lb/ft^3, *D1*=24_in, *hL*=2.0_ft^2/s^2, *W*=25_hp, *v1*=100_ft / s.

Solution: *Q*=18849.5559_ft^3/min, *M*=1206371.5790_lb/min, *ΔP*=-35_psi, *Δy*=100_ft, *v2*=93.1269_ft /s, *A1*=452.3893_in^2, *A2*=485.7773_in^2, *D2*=24.8699_in.

Flow in Full Pipes (3, 4)

These equations adapt Bernoulli's equation for flow in a round, full pipe, including power input (or output) and frictional losses. (See "FANNING" in Chapter 3.)



Equations:

$$\rho \cdot \left(\frac{\pi \cdot D^2}{4}\right) \cdot vavg \cdot \left(\frac{\Delta P}{\rho} + g \cdot \Delta y + vavg^2 \cdot \left(2 \cdot f \cdot \left(\frac{L}{D}\right) + \frac{\Sigma K}{2}\right)\right) = W$$

$$\Delta P = P2 - P1 \qquad \Delta y = y2 - y1 \qquad M = \rho \cdot Q$$

$$Q = A \cdot vavg \qquad A = \frac{\pi \cdot D^2}{4} \qquad Re = \frac{D \cdot vavg \cdot \rho}{\mu} \qquad n = \frac{\mu}{\rho}$$

Example:

Given: ρ=62.4_lb/ft^3, *D*=12_in, *vavg*= 8_ft/s, *P2*=15_psi, *P1*=20_psi, *y2*=40_ft, *y1*=0_ft, *μ*=0.00002_lbf*s/ft^2, *ΣK*=2.25, *ϵ*=0.02_in, *L*=250_ft.

Solution: *ΔP*=-5_psi, *Δy*=40_ft, *A*=113.0973_in^2, *n*=1.0312_ft^2/s, *Q*=376.9911_ft^3/min, *M*=23524.2358_lb/min, *W*=25.8897_hp, *Re*=775780.5.

Forces and Energy (4)

Variable	Description
α	Angular acceleration
ω	Angular acceleration
ωi,ωf	Initial and final angular velocities
ρ	Fluid density
τ	Torque
Θ	Angular displacement
α	Acceleration

Motion (8)

Variable	Description
α	Angular acceleration
ω	Angular velocity (Circular Motion), or Angular velocity at t (Angular Motion)
$\omega 0$	Initial angular velocity
ρ	Fluid density
θ	Angular position at t
$\theta 0$	Initial angular position (Angular Motion), or Initial vertical angel (Projectile Motion)
a	Acceleration
A	Projected horizontal area
ar	Centripetal acceleration at r
Cd	Drag coefficient
m	Mass
M	Planet mass
N	Rotational speed
R	Horizontal range (Projectile Motion), or Planet radius (Escape Velocity)
r	Radius
t	Time
v	Velocity at t (linear Motion), or Tangential velocity at r (Circular Motion), or Terminal velocity (Terminal Velocity), or Escape velocity (Escape Velocity)
$v0$	Initial velocity
vx	Horizontal component of velocity at t
vy	Vertical component of velocity at t
x	Horizontal position at t
$x0$	Initial horizontal position
y	Vertical position at t
$y0$	Initial vertical position

Reference: 3.

Angular Mechanics (4, 2)

Equations:

$$\tau = I \cdot \alpha \qquad Ki = \frac{1}{2} \cdot I \cdot \omega i^2 \qquad Kf = \frac{1}{2} \cdot I \cdot \omega f^2 \qquad W = r \cdot \Theta$$

$$W = Kf - Ki \qquad P = \tau \cdot \omega \qquad Pavg = \frac{W}{t} \qquad \omega f = \omega i + \alpha \cdot t$$

$$at = \alpha \cdot r \qquad \omega = 2 \cdot \pi \cdot N \qquad \omega i = 2 \cdot \pi \cdot Ni \qquad \omega f = 2 \cdot \pi \cdot Nf$$

Example:

Given: I=1750_lb*in^2, Θ=360_°, r=3.5_in, α=10.5_r/min^2, ωi=0_r / s.

Solution: r=1.1017E–3_ft_*lbf, Ki=0_ft*lbf, W=6.9221E–3_ft*lbf, Kf=6.9221E–3_ft*lbf, at=8.5069E–4_ft/s^2, Ni=0_rpm, ωf=11.4868_r/min, t=1.0940_min, Nf=1.8282_rpm, $Pavg$=1.9174E–7_hp.

Centripetal Force (4, 3)

Equations:

$$F = m \cdot \omega^2 \cdot r \qquad \omega = \frac{v}{r} \qquad ar = \frac{v^2}{r} \qquad \omega = 2 \cdot \pi \cdot N$$

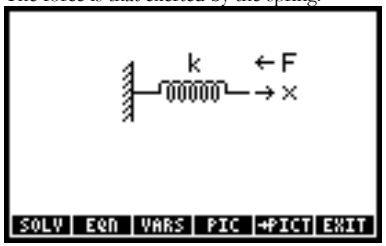
Example:

Given: m=1_kg, r=5_cm, N=2000_Hz.

Solution: ω=12566.3706_r/s, ar=7895683.5209_m/s, F=7895683.5209_N, v=628.3185_m/s.

Hooke's Law (4, 4)

The force is that exerted by the spring.



Equations:

$$F = -k \cdot x \qquad W = \frac{-1}{2} \cdot k \cdot x^2$$

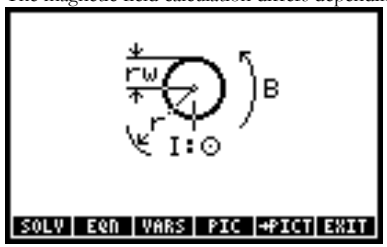
Example:

Given: k=1725_lbf/in, x=125_in.

Solution: F=-2156.25_lbf, W=-112.3047_ft*lbf.

Straight Wire (7, 1)

The magnetic field calculation differs depending upon whether the point is inside or outside the wire.



Equation:

$$B = \frac{\mu 0 \cdot \mu r \cdot I}{2 \cdot \pi \cdot r}$$

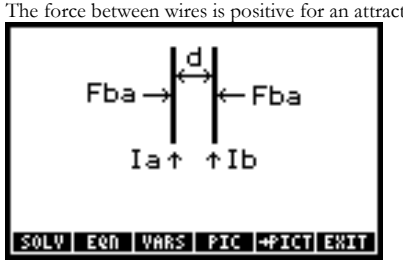
Example:

Given: μr=1, rw= 0.25_cm, r=0.2_cm, I=25_A.

Solution: B= 0.0016_T.

Force between Wires (7, 2)

The force between wires is positive for an attractive force (for currents having the same sign).



Equation:

$$Fba = \frac{\mu 0 \cdot \mu r \cdot L \cdot Ib \cdot Ia}{2 \cdot \pi \cdot d}$$

Example:

Given: Ia=10_A, Ib=20_A, μr=1, L=50_cm, d=1_cm.

Solution: Fba=2.0000E–3_N.

Gases (5)

Variable	Description
λ	Mean free path
ρ	Flow density
$\rho 0$	Stagnation density
A	Flow area
At	Throat area
d	Molecular diameter
k	Specific heat ratio
M	Mach number
m	Mass
MW	Molecular weight
n	Number of moles, or Polytropic constant (Polytropic Processes)
P	Pressure, or Flow pressure (Isentropic Flow)
$P0$	Stagnation pressure
Pc	Pseudocritical pressure
Pi, Pf	Initial and final pressures
T	Temperature, or Flow temperature (Isentropic Flow)
T0	Stagnation temperature
Tc	Pseudocritical temperature
Ti, Tf	Initial and final temperature
V	Volume
Vi, Vf	Initial and final volumes
$vrms$	Root-mean-square (rms) velocity
W	Work

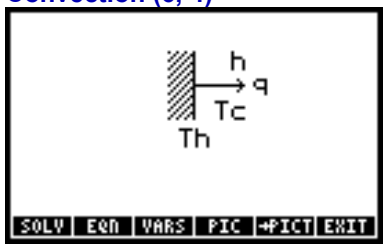
References:1, 3.

Ideal Gas Law (5, 1)

Equations:

$$P \cdot V = n \cdot R \cdot T \qquad m = n \cdot MW$$

Convection (6, 4)



Equations:

$$q = h \cdot A \cdot \Delta T \qquad q = h \cdot A \cdot (Th - Tc)$$

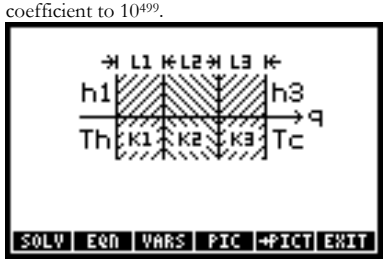
Example:

Given: *Tc*= 300_K, *A*=200_m^2, *h*=0.005_W/(m^2*K), *q*=10_W.

Solution: ⊿*T*=10_°C, *Th*=36.8500_°C.

Conduction + Convection (6, 5)

If you have fewer than three layers, give the extra layers a zero thickness and any nonzero conductivity. The two temperatures are fluid temperatures – if instead you know a *surface* temperature, set the corresponding convective coefficient to 10^{499}.



Equations:

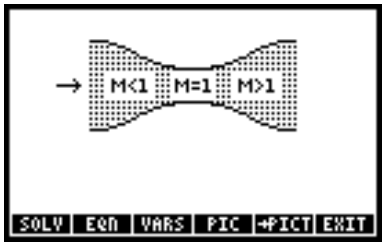
$$q = \frac{A \cdot \Delta T}{\frac{1}{h1} + \frac{L1}{k1} + \frac{L2}{k2} + \frac{L3}{k3} + \frac{1}{h3}} \qquad q = \frac{A \cdot (Th - Tc)}{\frac{1}{h1} + \frac{L1}{k1} + \frac{L2}{k2} + \frac{L3}{k3} + \frac{1}{h3}}$$

$$U = \frac{q}{A \cdot \Delta T} \qquad U = \frac{q}{A \cdot (Th - Tc)}$$

Example:

Given: ⊿*T*=35_°C, *Th*=55_°C, *A*=10_m^2, *h1*=0.05_W/(m^2*K), *h3*=0.05_W/(m^2*K), *L1*=3_cm, *L2*=5_cm, *L3*=3_cm, *k1*=0.1_W/ (m*K), *k2*=.5_W/ (m*K), *k3*=0.1_W/ (m*K).

Solution: *Tc*=20_°C, *U*=0.0246_W/ (m^2*K), *q*=8.5995_W.



Equations:

$$\frac{T}{T0} = \frac{2}{2 + (k-1) \cdot M^2} \qquad \frac{P}{P0} = \left(\frac{T}{T0}\right)^{\frac{k}{k-1}}$$

$$\frac{\rho}{\rho 0} = \left(\frac{T}{T0}\right)^{\frac{1}{k-1}}$$

$$\frac{A}{At} = \frac{1}{M} \cdot \left(\frac{2}{k+1} \cdot \left(1 + \frac{k-1}{2} \cdot M^2\right)\right)^{\frac{k+1}{2 \cdot (k-1)}}$$

Example:

Given: *k*=2, *M*=.9, *T0*=26.85_°C, *T*=373.15_K, *ρ0*=100_kg/m^3, *P0*=100_kPa, *A*=1_cm^2.

Solution: *P*=464.1152_kPa, *At*=0.9928_cm^2, *ρ*=215.4333_kg/m^3.

Real Gas Law (5, 6)

These equations adapt the ideal gas law to emulate real-gas behavior. (See "ZFACTOR" in Chapter 3.)

Equations:

$$P \cdot V = n \cdot Z \cdot R \cdot T \qquad m = n \cdot MW$$

Example:

Given: *Pc*=48_atm, *Tc*=298_K, *P*=5_kPa, *V*=10_l, *MW*=64_g/gmol, *T*=75_°C.

Solution: *n*=0.0173_gmol, *m*=1.1057E–3_kg.

Real Gas State Change (5, 7)

This equation adapts the ideal gas state-change equation to emulate real-gas behavior. (See "ZFACTOR" in Chapter 3.)

Equation:

$$\frac{Pf \cdot Vf}{Zf \cdot Tf} = \frac{Pi \cdot Vi}{Zi \cdot Ti}$$

Example:

Given: *Pc*=48_atm, *Pi*=100_kPa, *Pf*=50_kPa, *Ti*=75_°C, *Tc*=298_K, *Vi*=10_l, *Tf*=250_°C.

(Remember *Zf* and *Zi* are automatically calculated using these variables)

Solution: *Vf*=30.1703_l.

A Local label is a label that is only accessible in a local section like local variables in Pascal or C.
A local section starts at the beginning of a source, after a global label or after a link (see link section).
A local section finishes at the end of a source, before a link or before a global label.
A local label is identified by a '.' as the first character.

- Link labels
  A link label is a label that exists only in the link where it is declared, like a private clause in Object Pascal.
  A link label is identified by a '...' as the first character.

Notes:

1. In projects, using fewer global labels is better because a global label takes longer to compile and because it gives a better program structure. A good habit is to use global labels to cut the program in subroutines, and to use local labels inside these subroutines.
2. The command line editor is able to find labels in a source. See the GOTO selection in the command line TOOL menu.
3. Labels in System RPL should only be used by people who know what they are doing. They are only used for fixed address programs (absolute mode) which is pretty advanced programming.
4. Labels can not be given the same name as constants.

"extable"

"extable" is an external library that contains a list of constants. This list can be used by MASD as a basic list of constants and is especially useful to the System RPL programmer as most entry points are defined there (like TURNMENUOFF for example). In addition, it also contains a set of supported constants and ASM entry points for the ASM programmer. Please read the extable section in this document to find more information about this library.

Constants

Constants are a way for the user to associate a value to an alphanumerical name. This is extremely useful as it makes programs much easier to read and makes them more portable. One of the most popular ways to use constants is to represent memory address for storage of variables.

For example, instead of typing `D1=80100` every time it is needed, it is better to declare `DC Result 80100` at the beginning of the project and then to type `D1=(5)Result` when needed (it is more portable, more readable and less likely to cause errors).

You can create a constant in ASM or ARM mode by doing:

```
DC CstName ExpressionHex or
DEFINE CstName ExpressionHex or
EQU CstName ExpressionHex
```

In RPL mode, the only valid way to define a constant is:

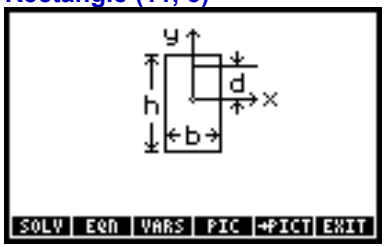
```
EQU CstName ExpressionHex
```

`ExpressionHex` is either a hexadecimal number or an expression (starting with a char that can not be confused with the start of a hex number (`0...9, A...F`). An expression starting with a hexadecimal number can be typed with a leading `$`, an expression starting with a decimal number can be typed with a leading `#` character. For an expression starting with a constant that starts with a `A...9` or `A...F` character, you should put the constant in brackets.

Notes:

1. A constant cannot be given the same name as a label.
2. The name of a constant follows the same rules as the name of a label.

Rectangle (11, 3)



Equations:

$$A = b \cdot h \qquad P = 2 \cdot b + 2 \cdot h \qquad I = \frac{b \cdot h^3}{12}$$

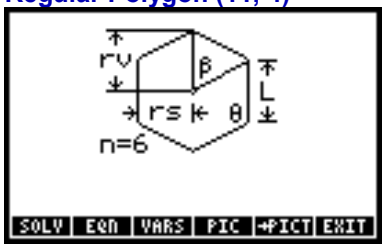
$$J = \frac{b \cdot h}{12} \cdot (b^2 + h^2) \qquad Id = I + A \cdot d^2$$

Example:

Given: *b*=4_chain, *h*=7_rd, *d*=39.26_in. Set guesses for *I*, *J*, and *Id* in km^4.

Solution: *A*=28328108.2691_cm^2, *P*=23134.3662_cm, *I*=2.9257E–7_km^4, *J*=1.8211E–6_km^4, *Id*=2.9539E–7_km^4.

Regular Polygon (11, 4)



Equations:

$$A = \frac{\frac{1}{4} \cdot n \cdot L^2}{TAN\left(\frac{180}{n}\right)} \qquad P = n \cdot L \qquad rs = \frac{\frac{L}{2}}{TAN\left(\frac{180}{n}\right)}$$

$$rv = \frac{\frac{L}{2}}{SIN\left(\frac{180}{n}\right)} \qquad \theta = \frac{n-2}{n} \cdot 180 \qquad \beta = \frac{360}{n}$$

Example:

Given: *n*=8, *L*= 0.5_yd.

Solution: *A*=10092.9501_cm^2, *P*=365.7600_cm, *rs*=55.1889_cm, *rv*=59.7361, θ=135_°, β=45_°.

Directives change the way MASD interprets your source. Theses directives begin with a ! and will be explained later.

Errors

If MASD detects one or more syntax error, it will push a list describing all errors on the stack. The ER command can help you make sense of that list, point you on the errors and let you correct them.

MASD will report a maximum of 16 errors before stopping compilation.

The ER command takes 2 objects as arguments:
The original source code (level 2)
The error list generated by MASD (level 1)

Normally, you should compile using a process similar to: IFERR ASM THEN ER END (this is what the ASM2 command does, by the way). Most people will just type the ASM command followed, if error, by the ER command.

Format of the error list

It's a list of at most 16 sub-lists. Each sub-list contains 3 system-binary and 1 global-name. The first system binary is an error message number. The second is an extra system binary used to indicate how 'too long' a jump is. The third one is the position in the source where the error is. The global name is either a NULLNAME if the error was in the main source or the filename of the buggy source.

Error messages

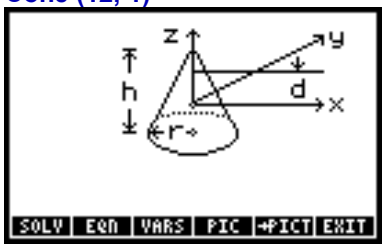
Message	Description
Invalid File	The file is not a valid source or macro. (must end with a @)
Too many	You can not do this operation as you are limited to a certain amount of them (for example, you can not have more than 64 simultaneous skips)
Unknown Instruction	Unknown instruction
Invalid Field	Incorrect field
Val betw 0-15 expected	An integer between 0 and 15 is expected
Val betw 1-16 expected	An integer between 1 and 16 is expected
Val betw 1-8 expected	An integer between 1 and 8 is expected
Label Expected	A label is expected
Hexa Expected	An hexadecimal number is expected
Decimal Expected	An decimal number is expected
Can't find	This object can not be located
Label already defined	This name is already in use
{ expected	A { character was expected
} expected	A } character was expected (this can happen if you do not close all the open skips for example)
(expected	A (character was expected
[or] expected	A [or] character was expected
Forbidden	This can not be done
Bad Expression	This expression is invalid
Jump too long	This jump is above the limit of the instruction (use a different type of jump)
Insuffisant Memory	There is not enough memory to compile
Matrix Error	You can not do this thing here because you are creating a matrix object

Solid Geometry (12)

Variable	Description
A	Total surface area
b	Base length
d	Distance to rotation axis in z direction
h	Height in z direction (Cone, Cylinder), or Height in y direction (Parallelepiped)
I, Ixx	Moment of inertia about x axis
Id	Moment of inertia in x direction at d
Izz	Moment of inertia about z axis
m	Mass
r	Radius
t	Thickness in z direction
V	Volume

Reference: 4.

Cone (12, 1)



Equations:

$$V = \frac{\pi}{3} \cdot r^2 \cdot h \qquad A = \pi \cdot r^2 + \pi \cdot r \cdot \sqrt{r^2 + h^2}$$

$$Ixx = \frac{3}{20} \cdot m \cdot r^2 + \frac{3}{80} \cdot m \cdot h^2 \qquad Izz = \frac{3}{10} \cdot m \cdot r^2$$

$$Id = Ixx + m \cdot d^2$$

Example:

Given: r=7_cm, b=12.5_cm, m=12.25_kg, d=3.5_cm.

Solution: V=641.4085_cm^3, A=468.9953_cm^2, Izz=0.0162_kg*m^2, Izz=0.0180_ kg*m^2, Id=0.0312_ kg*m^2.

Extension program

It is possible to enhance some of the statistics menus using a user library. The calculator does not provide every possible function in every area, but they let you customize the built in menu in order to add your functions as if they were built in.

Example: Customize the main statistic menu.

Ensure you are in RPL mode (MODE, +/−, ENTER) and attach the development library (256 ATTACH).

In a directory, create the following variables:

$ROMID 1324
$CONFIG 1
$TITLE "Statistic enhancements"
$VISIBLE { ABOUT }
$HIDDEN{ MessageHandler }
$EXTPRG 'MessageHandler'
ABOUT "This library is a statistic enhancement example"
MessageHandler

```
«
  IF DUP 1 R~SB ==
  THEN
    SWAP
    { { "7.New entry" « "My Stats" 1 DISP 7 * FREEZE » } } +
    SWAP
  END
»
```

Create the library (CRLIB) and store it in an extension port (0 STO). Now, run the statistic menu (↱ 5)!

How does it work? Each time the stat menu pops up, the calculator executes every extension program of every library in the system. This extension program takes on the stack a message number (and leaves it on the stack!). Each message number has a specific meaning as described below.

Here are the expected inputs and outputs for the extension program for different menus:

APPS menu
Input: { { "String" Action } …} ZERO
Output: Modified list ZERO

Main Statistics menu
Input: { { "String" Action } …} ONE
Output: Modified list ONE

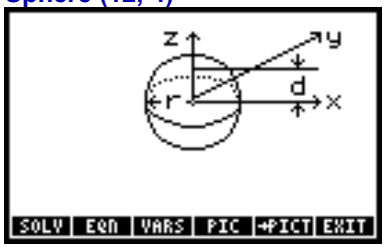
Hypothesis statistics menu
Input: { { "String" Action } …} TWO
Output: Modified list TWO

Confidence interval statistics menu
Input: { { "String" Action } …} THREE
Output: Modified list THREE

Finance menu
Input: { { "String" Action } …} FOUR
Output: Modified list FOUR

Numeric solver menu
Input: { { "String" Action } …} FIVE
Output: Modified list FIVE

Sphere (12, 4)



Equations:

$$V = \frac{4}{3} \cdot \pi \cdot r^3 \qquad A = 4 \cdot \pi \cdot r^2 \qquad I = \frac{2}{5} \cdot m \cdot r^2 \qquad Id = I + m \cdot d^2$$

Example:

Given: *d*=14_cm, *m*=3.75_kg, *Id*=486.5_lb*in^2.

Solution: *r*=21.4273_cm, *V*=41208.7268_cm^3, *A*=5769.5719_cm^2, *I*=0.0689_kg*m^2.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#*n*	→	¤*n*
¤*n*	→	#*n*

Example: #EC2Ah SB~B returns ¤ EC2Ah.

SERIAL

Description: Serial number command: Retrieves the calculator serial number. This is the software serial number and does not match the hardware serial number printed on the back of the calculator, but it is typically similar.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*"string"*

Example: SERIAL may return "HP50 Serial Number: CNA6110007".

S→H

Description: String to hex command: Returns the hex representation of the characters of a string.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"string$_1$"	→	*"string$_2$"*

Example: "A" S→H returns "14".

S~N

Description: String to name conversion command: Converts a string to a name and a name to a string. This command allows you to create invalid names, such as a null name.

Do not purge or move the null directory in HOME. Do not modify data in this directory.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"string"	→	*'global'*
'global'	→	*"string"*

Example: "" S~N returns ''.

SREV

Description: String reverse command: Gives the mirror image of a string.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"string$_1$"	→	*"string$_2$"*

Example: "ABC" SREV returns "CBA".

→S2

Description: Decompilation command: Decompiles an object in System RPL mode. If the "extable" library is installed, it will use the table to lookup the proper mnemonics for all known System RPL commands.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	*"string"*

Variable	Description
Js	Saturation current density
L	Drawn mask length (PN Step Junctions), or Drawn gate length (NMOS Transistors), or Channel length (JFETs)
Le	Effectives gate length
NA	P-side doping (PN Step Junctions), or Substrate doping (NMOS Transistors)
ND	N-side doping (PN Step Junctions), or N-channel doping (JFETs)
T	Temperature
tox	Gate silicon dioxide thickness
Va	Applied voltage
VBC	Base-to-collector voltage
VBE	Base-to-emitter voltage
Vbi	Built-in voltage
VBS	Substrate voltage
VCEsat	Collector-to-emitter saturation voltage
VDS	Applied drain voltage
VDsat	Saturation voltage
VGS	Applied gate voltage
Vt	Threshold voltage
Vt0	Threshold voltage (at zero substrate voltage)
W	Drawn mask width (PN Step Junctions), or Drawn width (NMOS Transistors), or Channel width (JFETs)
We	Effective width
xd	Depletion-region width
xdmax	Depletion-layer width
xj	Junction depth

References: 5, 8.

Example: 32.5 LR~R returns 3.25E1.

→LST

Description: Create symbolic command: This is equivalent to the RPL →LIST command, but it can also convert a program or symbolic to a list.

Input/Output:

$Level_{n+1}$/$Argument_1$ …$Level_2$/$Argument_n$	$Level_1$/$Argument_{n+1}$		Level 1/Item 1
obj_1 … obj_n	n	→	{obj_1, ...,obj_n}
	obj_1, ...,obj_n	→	{obj_1, ...,obj_n}

Example: « 3 2 + » →LST returns { « 3 2 + » }.

MAKESTR

Description: String creation command: Creates a test string of the given size.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	"string"

Example: 10 MAKESTR returns "ABCDEFG↵AB".

PEEK

Description: Memory read command: Reads nibbles from a specified address in memory.

Due to bank switching, the data read from address #40000h to #7FFFFh may not be accurate. Level 2 must contain the address to read and level 1 must contain the number of nibbles to read.

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
#n_1	#n_2	→	"string"

Example: #80711h #10h PEEK returns a string with sixteen hex characters.

PEEKARM

Description: Memory read command: Reads nibbles from a specified address in memory in the ARM address space.

Level 2 must contain the address to read and level 1 must contain the number of bytes to read.

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
#n_1	#n_2	→	"string"

Example: #7300000h #4h PEEKARM returns a string with eight hex characters.

POKE

Description: Memory write command: Writes nibbles to a specified address in memory.

You can not write data in the Flash ROM using this command.

Writing data in memory randomly can cause all memory to be lost.

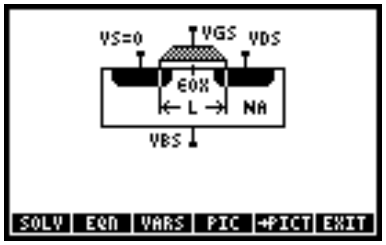
Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"string"	#n	→	

Example: "8" #100h POKE writes the hex digit 8 to memory address 0x100.

NMOS Transistors (13, 2)

These equations for a silicon NMOS transistor use a two-port network model. They include linear and nonlinear regions in the device characteristics and are based on a gradual-channel approximation (the electric fields in the direction of current flow are small compared to those perpendicular to the flow). The drain current and transconductance calculations differ depending on whether the transistor is in the linear, saturated, or cutoff region. The equations assume the physical geometry of the device is a rectangle, second-order length-parameter effects are negligible, shot-channel, hot-carrier, and velocity-saturation effects are negligible, and subthreshold currents are negligible. (See "SIDENS" in Chapter 3.)



Equations:

$$We = W - 2\cdot\Delta W \qquad Le = L - 2\cdot\Delta L \qquad Cox = \frac{\varepsilon ox\cdot\varepsilon 0}{tox}$$

$$IDS = Cox\cdot\mu n\cdot\left(\frac{We}{Le}\right)\cdot\left((VGS - Vt)\cdot VDS - \frac{VDS^2}{2}\right)\cdot(1+\lambda\cdot VDS)$$

$$\gamma = \frac{\sqrt{2\cdot\varepsilon si\cdot\varepsilon 0\cdot q\cdot NA}}{Cox}$$

$$Vt = Vt0 + \gamma\cdot\left(\sqrt{2\cdot ABS(\phi p) - ABS(VBS)} - \sqrt{2\cdot ABS(\phi p)}\right)$$

$$\phi p = \frac{-k\cdot T}{q}\cdot LN\left(\frac{NA}{ni}\right) \qquad gds = IDS\cdot\lambda$$

$$gm = \sqrt{Cox\cdot\mu m\cdot\left(\frac{We}{Le}\right)\cdot(1+\lambda\cdot VDS)\cdot 2\cdot IDS}$$

$$VDsat = VGS - Vt$$

Example:

Given: *tox*=700_Å, *NA*=1E15_1/cm^3, μn=600_cm^2/ (V*s), *T*=26.85_°C, *Vt0*=0.75_V, *VGS*=5_V, *VBS*=0_V, *VDS*=5_V, *W*=25_μ, ⊿*W*=1_μ, L=4_m, ⊿*L*=0.75_μ, λ=0.05_1/V.

Solution: *We*=23_μ, *Le*=2.5_μ, *Cox*=49330.4750_pF/cm^2, *γ*=0.3725_V^.5, *φp*= -.2898_V, *Vt*=0.75_V, *VDsat*=4.25_V, *IDS*=3.0741_mA, *gds*=1.5370E–4_S, *gm*=1.4466_mA/V.

→CD

Description: Hex to code command: Returns the code (Assembly program) object represented by an hex string.

A hex string is a string that only contains the characters '0' to '9' and 'A' to 'F'.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
Code	→	*"string"*

Example: "8F5076213016414 68..." →CD returns Code.

COMP→

Description: Composite out command: This is equivalent to the RPL LIST→ command, but it also works on Program and Symbolic objects.

Input/Output:

Level 1/Argument 1		Level$_{n+1}$/Item$_1$ ...	Level$_2$/Item$_n$	Level$_1$/Item$_{n+1}$
obj$_1$, ...,obj$_n$	→	*obj$_1$ ...*	*obj$_n$*	*n*

Example: « 3 2 + » COMP→ returns «, 3, 2, +, », 5.

CRC

Description: CRC computation command: Gives the CRC of a library or a string as a system binary.

This command gives you the CRC of the data in a library object or string (the CRC computation starts on the size of the object and finishes 4 nibbles before the end of the object).

Input/Output:

Level 1/Argument 1		Level 1/Item 1
Library Object	→	#*n*
"string"	→	#*n*

Example: "D9D20003621113" CRC returns # 8B63h.

CRLIB

Description: Create library command: Creates a library based on the variables in the current directory.

See the next section for details on the variables that must exist.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*Library n*

Example: CRLIB may return Library 902: LONGFLOAT.

ER

Description: Error checker command: Starts an interactive error-checking session. Takes the output of a failed run of ASM as input.

Input/Output:

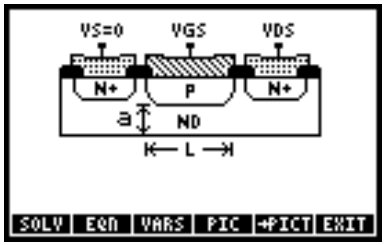
Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"*string*"	{*error list*}	→	"*edited string*"

H→

Description: Hex to object command: Returns the object represented by a hex string.

A hex string is a string that only contains the characters '0' to '9' and 'A' to 'F'.

If the string does not represent a valid object, this can corrupt your memory.



Equations:

$$Vbi = \frac{k \cdot T}{q} \cdot LN\left(\frac{ND}{ni}\right)$$

$$xdmax = \sqrt{\frac{2 \cdot \varepsilon si \cdot \varepsilon 0}{q \cdot ND} \cdot (Vbi - VGS + VDS)}$$

$$G0 = q \cdot ND \cdot \mu n \cdot \left(\frac{a \cdot W}{L}\right)$$

$$ID = G0 \cdot \left(VDS - \left(\frac{2}{3} \cdot \sqrt{\frac{2 \cdot \varepsilon si \cdot \varepsilon 0}{q \cdot ND \cdot a^2}}\right) \left((Vbi - VGS + VDS)^{\frac{3}{2}} - (Vbi - VGS)^{\frac{3}{2}}\right)\right)$$

$$VDsat = \frac{q \cdot ND \cdot a^2}{2 \cdot \varepsilon si \cdot \varepsilon 0} - (Vbi - VGS) \qquad Vt = Vbi - \frac{q \cdot ND \cdot a^2}{2 \cdot \varepsilon si \cdot \varepsilon 0}$$

$$gm = G0 \cdot \left(1 - \sqrt{\frac{2 \cdot \varepsilon si \cdot \varepsilon 0}{q \cdot ND \cdot a^2} \cdot (Vbi - VGS)}\right)$$

Example:

Given: *ND*=1E16_1/cm^3, *W*=6_*μ*, *a*=1_*μ*, L=2_*μ*, *μn*= 1248_cm^2/ (V*s), *VGS*= -4_V, *VDS*=4_V, T=26.85°C.

Solution: *Vbi*=0.3493_V, *xdmax*=1.0479_μ, *G0*=5.9986E-4_S, *ID*=0.2268_mA, *VDsat*=3.2537_V, *Vt*= -7.2537_V, *gm*=0.1462_mA/V.

Development Library Command Reference

A→

Description: Address out command: Returns the object stored at a specific address.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#*n*	→	*obj*

Example: `#3A57Ch A→` returns `SIN`.

→A

Description: Get address command: Returns the address of an object.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	#*n*

Example: `« SIN » OBJ→ DROP →A` returns `#3A57Ch`.

A→H

Description: Address to string command: Returns the hex representation of an address (you can then use this with the POKE command).

The hex representation of an address is a 5 character string where the address is written backwards.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	*"string"*

Example: `#3A57Ch A→H` returns `"C75A3"`.

→ALG

Description: Create symbolic command: This is equivalent to the RPL →LIST command, but it creates a symbolic object.

This command will also convert a program or a list to a symbolic object.

Input/Output:

$Level_{n+1}$/$Argument_1$ …$Level_2$/$Argument_n$	$Level_1$/$Argument_{n+1}$		Level 1/Item 1
obj_1 … obj_n	*n*	→	*'symb'*
	{obj_1, ...,obj_n}	→	*'symb'*
	obj_1, ...,obj_n	→	*'symb'*

Example 1: `'X' 2 ^ 4 + 5. →ALG` returns `'X^2+4'`.

Example 2: `« 5 'A' * » →ALG` returns `'5*A'`.

APEEK

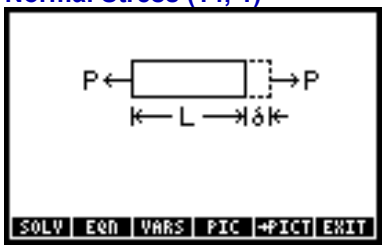
Description: Address PEEK command: Reads the address stored at an address.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#*n*	→	#*n*

Example: `#80711h APEEK` returns the address of the home directory.

Normal Stress (14, 1)



Equations:

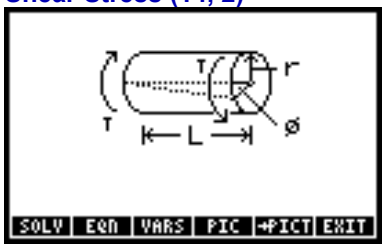
$$\sigma = E \cdot \epsilon \qquad \epsilon = \frac{\delta}{L} \qquad \sigma = \frac{P}{A}$$

Example:

Given: P=40000_lbf, L=1_ft, A=3.14159265359_in^2, E=10E6_psi,

Solution: δ=0.0153_in, ϵ=0.0013, σ=12732.3954_psi.

Shear Stress (14, 2)



Equations:

$$\tau = G \cdot \gamma \qquad \gamma = \frac{r \cdot \phi}{L} \qquad \tau = \frac{T \cdot r}{J}$$

Example:

Given: L=6_ft, r=2_in, J=10.4003897419_in^4, G=12000000_psi, τ=12000_psi.

Solution: T=5200.1949_ft*lbf, ϕ=2.0626_°, γ=5.7296E–2_°.

Stress on an Element (14, 3)

Stresses and strains are positive in the directions shown.

Waves (15)

Variable	Description
β	Sound level
λ	Wavelength
ω	Angular frequency
ρ	Density of medium
B	Bulk modulus of elasticity
f	Frequency
I	Sound intensity
k	Angular wave number
s	Longitudinal displacement at x and t
sm	Longitudinal amplitude
t	Time
v	Speed of sound in medium (Sound Waves), or Wave speed (Transverse Waves, Longitudinal Waves)
x	Position
y	Transverse displacement at x and t
ym	Transverse amplitude

Reference: 3.

Transverse Waves (15,1)

Equations:

$$y = ym \cdot SIN(k \cdot x - \omega \cdot t) \qquad v = \lambda \cdot f \qquad k = \frac{2 \cdot \pi}{\lambda} \qquad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: ym=6.37_cm, k=32.11_r/cm, x=.03_cm, ω=7000_r/s, t=1_s.

Solution: f= 1114.0846_Hz, λ=0.0020_cm, y=2.6655_cm, v=218.0006_cm/s.

Longitudinal Waves (15, 2)

Equations:

$$s = sm \cdot COS(k \cdot x - \omega \cdot t) \qquad v = \lambda \cdot f \qquad k = \frac{2 \cdot \pi}{\lambda} \qquad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: sm=6.37_cm, k=32.11_r/cm, x=0.03_cm, ω=7000_r/s, t=1_s.

Solution: s=5.7855_cm, v=2.1800_m/s, λ=0.1957_cm, f=1114.08456_Hz.

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Invalid N	Attempted to calculate *I%YR* with $N < 1$ or $N \geq 10^{10}$.	E604
Invalid Name	Received illegal filename, or server asked to send illegal filename.	C17
Invalid PPAR	*PPAR* not a list, or one or more objects in list missing or invalid.	12E
Invalid PRTPAR	*PRTPAR* not a list, or one or more objects in list missing or invalid.	C13
Invalid PTYPE	Plot type invalid for current equation.	620
Invalid PYR	*PYR* must be a positive real number.	E605
Invalid Repeat	Alarm repeat interval out of range.	D03
Invalid Server Cmd.	Invalid command received while in Server mode.	C08
Invalid Syntax	Unable execute ENTER, OBJ→, or STR→ due to invalid object syntax.	106
Invalid Time	Time argument not real number in correct format or out of range.	D02
Invalid Unit	Unit operation attempted with invalid or undefined user unit.	B01
Invalid User Function	Type or structure of object executed as user-defined function was incorrect.	103
Invalid Σ Data	Statistics command executed with invalid object stored in *ΣDAT*.	601
Invalid Σ Data LN(Neg)	Non-linear curve fit attempted when *ΣDAT* matrix contained a negative element.	605
Invalid Σ Data LN(0)	Non-linear curve fit attempted when *ΣDAT* matrix contained a 0 element.	606
Invalid ΣPAR	*ΣPAR* not list, or one or more objects in list missing or invalid.	604
Invalid #Periods	AMRT requires a positive integer number of periods.	E606

3. A constant value is stored in 16 nibbles.
4. Having constants starting with something that can be interpreted as a hex number, or an ARM register is not a good idea as the compiler might get confused. For example: `DC SPFOO 4 MOV R4 SPFOO` will generate an error on `FOO` as the compiler will interpret the mov as a mov from SP to R4.

MASD introduces a 'constant pointer' called CP which helps to define constants. CP is defined by:

```
CP=ExpressionHex
```

CP is defined by 5 nibbles; its default value is 80100 (an area of memory that can be used freely by programmers).

To declare a constant with the current CP value and then increase CP by Increment:

```
EQUCP Increment ConstantName
```

Notes:

1. In ASM and ARM mode, `DCCP Increment ConstantName` is also valid
2. Increment is a hexadecimal value, to use a decimal value, put a leading `#`.

For example, if CP equals to $10, the following defines a Foo constant with a value of $10 and then changes the value of CP to $15:

```
EQUCP 5 Foo
```

Several constants can be defined at once using CP.

```
: Inc CstName0 CstName1 ... CstNameN-1 :
```

The above defines *N* constants `CstNamex` with a value of CP+*x***Inc* and then changes the CP value to CP+*N***Inc*. By default, *Inc* is a decimal number or an expression that can be immediately evaluated.

These features are extremely useful to define areas of memory for storage of ASM program variables.

Notes:

1. If the entry point library (see related section) is installed on your calculator, all the values in the constant library will be available in your programs the same way than constants are.
2. You can define a constant in your program to override the value of an entry in the equation library.

Expressions

An expression is a mathematical operation that is calculated at compilation time. Terms of this operation are hexadecimal or decimal values, constants or labels. An expression stops on a separation character or a '`]`'.

```
DCCP 5 @Data
...
Di=(5)@Data+$10/#2
D0=(5)$5+DUP
LC(5)"DUP"+#5
```

The above are correct expressions (provided that the entry point library is installed).

Notes:

- A hexadecimal value must begin with a `$`.
- A decimal value may begin with a `#` or a number directly.
- A `&` or `(*)` equals the offset of the current instruction in the program (This value has no meaning in itself, but may be used to calculate the distance between a label and the current instruction). In absolute mode, this represents the final address of the instruction.
- The value of a label is the offset of the label in the program (This value has no meaning in itself, but may be used to calculate the distance between a label and the current instruction). In absolute mode, this represents the final address of the instruction.

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Copied to stack	EQUATION copied selected equation to stack.	623
Current equation:	Identifies current equation.	608
Deleting Column	Matrix Writer application is deleting a row.	504
Deleting Row	Name of existing directory variable used as argument.	503
Directory Not Allowed	Name of existing directory variable used as argument.	12A
Directory Recursion	Attempted to store a directory into itself.	002
Empty catalog	No data in current catalog (Equation, Statistics, Alarm)	60D
Empty stack	The stack contains no data.	C15
Enter alarm, press SET	Alarm entry prompt.	61A
Enter eqn, press NEW	Store new equation in *EQ*.	60A
Enter value (zoom out if >1), press ENTER	Zoom operations prompt.	622
EQ Invalid for MINIT	*EQ* must contain at least two equations (or programs) and two variables.	E403
Extremum	Result returned by HP Solve application or ROOT is an extremum rather than a root.	A06
HALT Not Allowed	A program containing HALT executed while Matrix Writer application, DRAW, or HP Solve application active.	126
I%YR/PYR ≤ -100	Interest per period must be greater than -100%	E603
Illegal During MROOT	Multiple-Equation Solver command attempted during MROOT execution.	E406

- In case of a list, only the first object of the list will be included following the previous rules

The syntax in ASM or ARM mode is: ⁄FileName

Note: To know how MASD looks for the FileName file, see the following section.

You can also include a complete object (prologue included) using INCLUDE or INCLOB. In ASM or ARM mode, use INCLUDE or INCLOB followed by a filename to include an object, in RPL mode, use INCLOB.

Filename conventions

MASD sometimes needs to find a file in the calculator's memory. The file can be found either by specifying the file name and location, or only the file name to be search in the directory search list.

The initial directory search list contains the current directory, and all parents directory up to the HOME directory.

You can add a directory in the directory search list using !PATH+ RepName where RepName identifies a directory name using filename rules.

To specify a full path, use

H⁄ to specify HOMEDIR as the root.

x⁄ where *x* is a port number, to specify a port as root. Note: you cannot use 3 (SD card) here.

This root is followed by a list of directories, ending with the name of the file.

2⁄FOO⁄BAR⁄BRA specifies the BRA file in the BAR directory, stored in the FOO backup of port 2.

H⁄ME⁄YOU specifies the YOU file in the ME directory, in the HOMEDIR.

Note: You cannot have more than 16 entries in the directory search path.

Compilation directive

The following instruction modifies the way MASD reacts and compiles things. They are valid in all modes:

Directive	Description
!PATH+ DirName	Add the specified directory in the search path list.
!NO CODE	MASD will not generate a $02DCC prologue but will directly output the data. If the generated file is not a valid object, an error will be generated.
!DBGON	MASD will generate code when DISP or DISPKEY are found in the source.
!DBGOFF	MASD will not generate code when DISP or DISPKEY are found in the source.
!1-16	Switch to 1-16 mode.
!1-15	Switch to 0-15 mode.
!RPL	Switch to RPL mode.
!ASM	Switch to ASM mode.
!ARM	Switch to ARM mode.
!FL=0.a	Clear the *a* compilation flag.
!FL=1.a	Set the *a* compilation flag.
!?FL=0.a	Compile the end of the line if flag *a* is set.
!?FL=1.a	Compile the end of the line if flag *a* is clear.
!ABSOLUT Addr	Switch to absolute mode. The program begins at the address *Addr*. Note: MASD always considers the prolog $02DCC and code length to be the beginning of the program even if !NO CODE is set.
!ABSADR Addr	If in absolute mode, add blank nibbles to continue at the specified address. If not possible, errors.

The Entry Point Library: Extable

The entry point library is an external library (you can get it from the HP web site) that contains a table of entry point names and addresses. This is used by the MASD compiler to get the value of System RPL entry points or assembler constants (like TURNMENUOFF for example).

This library should be stored in port 0, 1 or 2. If you want to program in System RPL, you must install this library. This library also contains 4 functions:

nop

Description: This function is here for internal purposes and should not be used. Running this function does nothing.

Input/Output: None

GETNAME

Description: Looks up the name of an entry from its address.

As multiple entries might have the same address, GETNAME is not a bijective (one-to-one) function.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n	→	*"string"*

Example: `#054AFh GETNAME` returns `"INNERCOMP"`.

GETADR

Description: Looks up the address of an entry from its name.

As multiple entries might have the same address, GETNAME is not a bijective (one-to-one) function.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"string"	→	*#n*

Example: `"INNERCOMP" GETADR` returns `#054AFh`.

GETNAMES

Description: Finds all the entries whose names start with a specific string.

Giving a null string as an input will return a list of all the entry points.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"string"	→	*{"string," ...,"string,"}*

Example: `"COMP" GETNAMES` returns `{ "COMPCONFCRC" "COMPEVAL" }`.

Library 257

Library 257 is used by the development library but is also not attached by default. It provides three commands. asm is identical to ASM in library 256 (ASM is a stub that calls asm), er is identical to ER in library 256 (ER is a stub that calls er), and ASM2 calls asm and automatically calls er if there are errors.

Saturn ASM mode

This section is only applicable to the Saturn ASM mode.

CPU architecture

This section's purpose is to make experienced ASM programmers familiar with the Saturn architecture, not to teach anyone to program in Saturn ASM.

The Saturn CPU has 12 main registers:

A, B, C, D, R0, R1, R2, R3 and R4 are 64 bits register (see description below),
D0 and D1 are 20 bits pointers (you can only access memory through them, the Saturn is a little endian),
PC, 20 bit program counter.

In addition, there are 16 flags ST0 to ST15 (12-15 being reserved for the system) that are 1 bit registers that are accessible separately, a carry that is set when operation overflow or tests are validated and can be tested using the GOC (Go On Carry) and GONC (Go On No Carry) jump instruction, a decimal/hexadecimal mode (SETHEX and SETDEC) that affects the way + and – instructions on the A, B, C and D register works (default Is HEX), and a 8 level return stack for GOSUBs (and RTN).

64 bit registers

Most operations on 64 bit registers will act on a specific "field". A field is a division in a 64 bit register.

If this represents the 16 nibbles of a 64 bit register, the fields cover the register as follows:

<table>
<tr><td>F</td><td>E</td><td>D</td><td>C</td><td>B</td><td>A</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td></td><td></td><td></td><td></td><td>P</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td colspan="12">WP</td></tr>
<tr><td>S</td><td colspan="12">M</td><td>XS</td><td colspan="2">B</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="4">A</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">X</td></tr>
</table>

The P field location depends of the value of the 4 bit P register (i.e. you can move it), and so does the WP field. Please look at the instruction set to see what instructions are available to the programmer. We will usually write "Rf" to indicate a register (uppercase) and a field (lowercase), as in `Am`.

In addition, in the new simulated Saturn architecture, 7 new fields F1 to F7 have been introduced. You can define the field mask by using the `SETFLDn` where n is a number between 1 and 7 to define the field Fn using the mask in Cw as in this example:

```
LC FF00000000000FF SETFLD1
LA 123456789ABCDEF0
LC 0FEDCBA987654321
A=A!C.F1
```

A is now equal to:

```
1F3456789ABCDEF1
```

ie: the instruction on F fields equate to:

reg1= (reg1 & ~ mask) | ((reg1 & mask) operation (reg2 & mask))

These new fields are available for all instructions that previously used the so called 'f' encoding and includes the following instructions:

Reg=Reg&Reg.f, Reg=Reg!Reg.f, DATx=Reg.f, Reg=DATx.f, Reg=Reg+Cte.f, Reg=Reg-Cte.f, RegSRB.f, RReg=Reg.f, Reg=RReg.f and RegRRegEX.f.

```
LDRB R6 [R1 2408]               % set the flag ST0
ORR R6 R6 1
STRB R6 [R1 2408]
LDMIA sp! (R4 R5 R6 R7 R8 PC)   % restore all registers and return

!ASM                            % back in ASM mode
*end
}
C=RSTK D0=C                     % D0 points to ARM instruction
D1=80100                        % D1 points at a place where
                                % I can copy the program
LC(5) end-start MOVEDN          % copy n nibbles

C=0.B SETLNED                   % hide the header

D1=8229E                        % point on 2Kb free memory
LC A9 A=0.W A-1.W { DAT1=A.W D1+16 C-1.B UPNC } % paint it in black
D0=00120 LC 8229E DAT0=C.A      % point the screen to that memory
D0=C                            % D0 points to that memory

LC FFFFEFFF D=C.W               % D=Y=-1
LC 4F R3=C.B                    % loop 80 times
{
  C=0.W LC 1800 C=-C.W B=C.W    % B=X=-1.5
  LC 82                         % loop 131 times
  A=0.S A+1.S                   % set bit 0 in As
  {
    RSTK=C                      % save loop counter in RSTK
    LC 80100 ARMSAT             % evaluate the ARM code
    ?ST=0.0                     % if point is in the set, do nothing
    {
      C=DAT0.S C-A.S DAT0=C.S   % else, turn the pixel off
    }

    A+A.S SKNC { D0+1 A+1.S }   % next pixel

    C=0.W LC 40 B+C.W           % increment X
    C=RSTK C-1.B UPNC           % count down and loop
  }
  D0+2                          % next graphic line
  C=0.W LC 66  D+C.W            % increment Y
  C=R3.W C-1.B R3=C.W UPNC      % count down and loop
}
LC FF OUT=C { C=IN2 ?C=0.B UP } % wait for no key down
{ C=IN2 ?C#0.B UP }             % Wait for 1 key down
INTON                           % restore the keyboard interrupt
LC 10 SETLNED                   % restore the header size
SCREEN CD0EX D0=00120 DAT0=C.A  % restore the screen pointer
LOADRPL                         % return to RPL
ENDCODE                         { end of ASM program }
;                               { end of RPL program }

@"
```


The foundation of Skips is the Block structure. A block is enclosed in { and }, and can be nested within another block. The following instructions deal with blocks.

SKIPS instructions	Equivalents
{ … }	Defines a block (generates no code)
SKIP { …}	GOTO .S … *.S
SKIPL { …}	GOTOL .S … *.S
SKIPC { …}	GOC .S … *.S
SKC {…}	GOC .S … *.S
SKIPNC {…}	GONC .S … *.S
SKNC { …}	GONC .S … *.S
Test SKIPYES {…}	Test GOYES .S … *.S
Test { …}	Test GOYES .S … *.S
Test { …}	/Test GOYES .S… *.S
Test -> { …}	/Test GOYES .S … *.S
SKUB { …}	GOSUB .S … *.S
SKUBL { …}	GOSUBL .S … *.S
STRING { …}	$/02A2C GOIN5 *.S … *.S (to create a character string)
CODE { …}	$/02DCC GOIN5 *.S … *.S (to create a code object)
STROBJ $PROLOG { …}	$(5)PROLOG GOIN5 .S … *.S (to create a 'prolog – length' object)

/Test is the opposite of Test. For example if Test is ?A<C.A, /Test is ?A>=C.A. The test instructions dealing with the hardware register (?HST=0, ?MP=0, ?SR=0, ?XM=0, ?SB=1, ?HST=1, ?MP=1, ?SR=1, ?XM=1 and ?SB=1) cannot be inverted.

Once blocks are defined, special instructions can be used in them. The instructions called EXIT and UP allow jumping to the end or to the beginning of a block.

These instructions	are equivalent to
{	*.Beginning
EXIT	GOTO.End
EXITC	GOC.End
EXITNC	GONC.End
?A=0.A EXIT	?A=0.A ".End
UP	GOTO.Beginning
UPC	GOC.Beginning
UPNC	GONC.Beginning
?A=0.A UP	?A=0.A ".Beginning
}	*.End

Note: In Saturn mode do not be confused between EXIT and UP instructions, which are GOTOs, and EXIT and UP after a test, which are GOYES's. EXIT and UP can jump to the beginning or to the end of an upper-level block by specifying the number of blocks to exit, after the UP or EXIT instructions.

Example of a Saturn assembly language program using the MASD compiler

```
"!NO CODE !RPL
* This program display a 131*64 graphic in a pretty way :-)
* DO LCD->, run it, and enjoy!
* This program has been created by Philippe Pamart
::
* remove the menu and test for a grob
CK1&Dispatch grob
::
TURNMENUOFF
CODE

% R0a: X
% R1a: Y
% R2a: @ grob

SAVE GOSBVL DisableIntr                 % No interrupts
A=DAT1.A D0=A LC 00014 A+C.A R2=A.A      % adr 1st pixels of the grob
D0+10 A=0.W A=DAT0.10 C=0.W LC 8300040 ?A=C.W % test the size
{ *.End GOSBVL AllowIntr LOADRPL }      % if not ok, return to RPL

GOSBVL "D0->Row1" D1=A D0-15 C=DAT0.A C-15.A GOSBVL WIPEOUT % erase screen
LC 0003F R1=C.W                         % initial position in Z

{
  LC 00082                              % we are ready to scan right to left
  {
    R0=C.A                              % save the counter
    LC 001 GOSBVL OUTCINRTN ?CBIT=1.6 -> .End % If backspace, then stop
    GOSUB .PointAndLine                 % test the current point
    C=R0.A C-1.A UPNC                   % go one pixel on the right
  }
  A=R1.W A-1.A R1=A.A                   % go one line higher
  {                                     % ready to scan from right to left
    LC 001 GOSBVL OUTCINRTN ?CBIT=1.6 -> .End % If backspace, then stop
    GOSUB .PointAndLine                 % test the current point
    A=R0.A A+1.A R0=A.A LC 83 ?A#C.B UP % go one pixel on the left
  }
  A=R1.A A-1.A R1=A.A UPNC              % go one line higher (if not finish)
}
GOTO .End


*.PointAndLine                          % This tests the current pix, returns
                                        % if the pixel is white, draw a line
                                        % if it is black
A=R1.A A+A.A C=R2.A C+A.A ASL.A A+C.A % A: @ line of pixel in the grob
C=R0.A P=C.0 CSRB.A CSRB.A A+C.A D0=A % D0: point on the pixel to test,
                                        % P = number of the pixel to test in
                                        % nibble (in Z/4Z)
LC 2481248124812481 P=0                 % Cp: pixel mask
A=DAT0.B A&C.P ?A=0.P RTY               % test the pixel. if white, return
GOSUB LIGNE GOSUB LIGNE                 % else, draw line twice in Xor mode
GOSBVL "D0->Row1" D0-20                 % and draw the pixel in black.
A=R0.A C=R1.A GOVLNG aPixonB

*LIGNE
GOSBVL "D0->Row1" D0-20                 % D0 point on the screen
A=R0.A B=A.A LA 00041                    % A/B: X coordinates
C=R1.A D=C.A C=0.A                        % C/D: Y coordinates
GOVLNG aLineXor                           % draw the line!

ENDCODE
;
;
@"
```


Saturn instructions syntax

In this section:

x is a decimal number between 1 and 16. An expression can be used if its value can be determined at the first encounter.
h is a hexadecimal digit.
a is a decimal number ranging from 1 to 16 or a 0 to 15 number depending of the current mode (0-15 or 1-16). An expression can be used, if its value can be determined at the first encounter.
f is a field A, B, X, XS, P, WP, M, S, F1, F2, F3, F4, F5, F6 or F7.
Reg is a working register A, B, C or D.
RReg is a scratch register R0, R1, R2, R3 or R4.
Exp is an expression.
Cst is a decimal constant. An expression can be used if its value can be determined at the first encounter.
DReg is a pointer register D0 or D1.
Data is memory data pointed by D0 or D1. It means `DAT0` or `DAT1`.

Note: For instructions that use two working registers, instruction using the pairs A-B, B-C, C-D and A-C are smaller and faster (if the `Fn` fields are not used).

For instructions like *Reg1=Reg1…* you can write only *Reg1…* Example: `A=A+C.A` is the same as `A+C.A`.

Syntax	Example	Notes
Reg=0.f	`A=0.M`	Sets the specific field of the register to 0
Reg=1.f	`A=1.M`	Sets the specific field of the register to 1
LC hhh..hhh LA hhh..hhh	`LC 80100` `LA #1024`	The number of nibbles loaded in the register is the number of characters necessary to write the value. So LC #12 will be equivalent to LC 00C. Note: the less significant nibble is loaded in the nibble P (as in the value of the register P) of the register, the next one into nibble p+1 mod 16, and etcetera.
LCASC(x) chrs LAASC(x) chrs	`LCASC(4) MASD` `LAASC(5) ROCKS`	Loads the hexadecimal value of *x* characters into C. *x* must be between 1 and 8. See note on LC instruction
LC(x) Exp LA(x) Exp	`LC(5)@Buf+Off`	Loads the result of an expression into C or A, using x nibbles. See note on LC instruction
Reg1=Reg2.f	`A=B.X`	Copies the value of a specific field of a register into the same field of another register
Reg1Reg2EX.f	`ABEX.W`	Exchanges the value of 2 registers on the given field. Note: this is not valid for the `Fn` fields
Reg1=Reg1+Reg2.f Reg1+Reg2.f	`A=A+B.A` `C+D.A`	Adds the value of the specific field of one register to the other register. Note: If Reg1 and Reg2 are the same, this is a multiply by 2 instruction Note: This instruction is affected by the DEC/HEX mode only if the field is not a F field and the registers are AB, BC, CD or AC.
Reg1=Reg1-Reg2.f Reg1-Reg2.f	`A=A-B.A` `C-D.A`	The following instructions are also available (but not on the `Fn` fields): `A=B-A.`*f* `B=C-B.`*f* `C=A-C.`*f* `D=C-D.`*f* see note on Reg1=Reg1+Reg2.f

Defines

If the instruction matches a define, the correct code is inserted (see the DEFINE instruction)

Labels

If the instruction matches the name of a constant or a label, the value of the said constant or label is inserted (if you insert a label, be sure to know what you are doing and to be in absolute mode).

extable

If the instruction matches an entry in the extable (see appropriate section at the end of this document) the value associated with this entry is used.

DUP Will produce 88130

Note: Using an external table is much faster than using constants. On the other hand, constants are project dependent, which is not the case of an external table.

Tokens

Then, the following instructions are tested:

Token	Description
::	Program prologue $02D9D
; or END	List, Program or Algebraic end $0312B
«	List prologue $02A74
»	List end $0312B
MATRIX	Algebraic matrix object
SYMBOLIC	Algebraic prologue $02AB8
UNIT	Unit prolog $02ADA
FPTR2 ^constant	Flash pointer from constant
FPTR bank value	Flash pointer from value
# cst	System Binary of *cst* value, given in hexadecimal. If there is no space between the # and the cst, MASD will try, if possible, to use the internally defined system binary
PTR cst	Address. PTR 4E2CF generates FC2E4.
ACPTR cst1 cst2	Extended pointer with given hexadecimal values for the address and switch address
ROMPTR2 ~xlib_name	XLIB object from constant
ROMPTR LibN ObjN	XLIB object from value
% real	Real number
%% real	Long real number
C% real1 real2	Complex number
C%% real1 real2	Long complex number
"..."	Character string. Special characters can be included by typing \ and the ASCII value on two hexadecimal characters. \ can be inserted by typing \\
ZINT decimalvalue	Integer
ID name	Global name (see " for info on character encoding)
LAM name	Local name (see " for info on character encoding)
TAG chrs	Tagged object

Syntax	Example	Notes
DReg=hh *DReg=hhhh* *DReg=hhhhh* *DReg=(2)Exp* *DReg=(4)Exp* *DReg=(5)Exp*	D0=AD D0=0100 D0=80100 D0=(2)label D0=(4)lab+$10 D1=(5)Variable	Change the first 2, 4 or all nibbles of the Data register with the given value
Dreg=Reg	D0=A	Reg can only be A or C
Dreg=RegS	D0=CS	Sets the first 4 nibbles of Dreg with the 4 first nibbles of Reg Reg can only be A or C
RegDRegEX	AD0EX	Reg can only be A or C
RegDRexXS	AD1XS	Exchange the first 4 nibbles of Dreg with the 4 first nibbles of Reg Reg can only be A or C
DReg=DReg+Cst *DReg+Cst* *DReg=DReg-Cst* *DReg-Cst*	D0=D0+12 D1+25 D1=D1-12 D1-5	Note 1: The Saturn processor is not able to add a constant greater than 16 to a register, but if *cst* is greater than 16, MASD will generate as many instructions as needed. Note 2: Even if adding constants to a register is very useful, big constants should be avoided because they will slow down execution, and generate a big program. Note 3: After adding a constant greater than 16, the carry should not be tested. Note 4: You can put an expression instead of the constant (MASD must be able to evaluate the expression right away). If the expression is negative, MASD will invert the addition to a subtraction and vice versa. Note 5: Be careful when using subtraction; it's easy to be misled. D0-5-6.A is equivalent to D0+1.A, not D0-11.A

Please read the section on test above for information on what MUST follow a test instruction.
f can NOT be a Fn field.

?Reg1=Reg2.f	?A=C.B	
?Reg1#Reg2.f	?A#C.A	The HP ≠ character can also be used
?Reg=0.f	?A=0.B	
?Reg#0.f	?A#0.A	The HP ≠ character can also be used
?Reg1<Reg2.f	?A<B.X	
?Reg1>Reg2.f	?C>D.W	
?Reg1<=Reg2.f	?A<=B.X	The HP ≤ character can be used
?Reg1>=Reg2.f	?C>=D.W	The HP ≥ character can be used
?RegBIT=0.a ?RegBIT=1.a	?ABIT=0.5 ?ABIT=1.number	Test if a specific bit of A or C register is 0 or 1 Reg must be A or C
	A=PC C=PC	Sets Aa or Ca to the address of the next instruction
	PC=A PC=C	Set PC to the value contained in Aa or Ca
	APCEX CPCEX	Exchange the value of PC with register Aa or Ca
	PC=(A) PC=(C)	Sets PC to the value read at the address contained in Aa or Ca
	SB=0 XM=0 SR=0 MP=0 HST=0.a ?SB=0 ?XM=0 ?SR=0 ?MP=0 ?HST=0.a	SB, XM, SR and MP are 4 bits in the HST register. They can be set to 0 by the specific instruction and tested. SB is set to 1 by RegSR and RegSRB instruction, XM by RTNSXM instruction and SR and MP should always be 0 (hardware related stuff). HST=a sets all bits set to 1 in a to 0 in the HST register. ?HST=a test that all bits set to 1 in a are 0 in the HST register

2332	REG C;
2340	REG D;
2348	REG R0;
2356	REG R1;
2364	REG R2;
2372	REG R3;
2380	REG R4;
2388	U32 D0
2392	U32 D1;
2396	U32 P, P4, P4_32; // P4 = 4*P, P4_32 = 4*P-32, use setP() to modify P.
2408	U32 ST;
2412	U32 HST;
2416	U32 carry; // 0 or !0
3420	BOOL dec; // 0→hex or 1→dec
	U32 RSTK[NB_RSTK];
	U32 RSTK_i; // Index for next push.
	REG FIELD[32]; // Field masks.
	U32 FIELD_START[32]; // Lowest nibble of the field.
	U32 FIELD_LENGTH[32]; // Length of the field.

Therefore, `LDR R2 [R1 2316]` allows you to read the lower 32 bits of the Saturn register A.

```
LDR R2 [R1 1]
LDR R3 [R2 1]
```
allows you to read the first 8 nibbles at Saturn address 01008

The following file can be used to declare your Saturn chipset structure.

```
"!ASM
CP=0
DCCP #1028 SREAD
DCCP #1028 SWRITE
DCCP #260 SPRIORITY
DCCP 8 SRA
DCCP 8 SRB
DCCP 8 SRC
DCCP 8 SRD
DCCP 8 SR0
DCCP 8 SR1
DCCP 8 SR2
DCCP 8 SR3
DCCP 8 SR4
DCCP 4 SD0
DCCP 4 SD1
DCCP 4 SRP
DCCP 4 SRP4
DCCP 4 SRP32
DCCP 4 SST
DCCP 4 SHST
DCCP 4 SCARRY
DCCP 4 SDEC
DCCP #32 SRSCK
DCCP 4 SRSTKP
DCCP #256 SFMASK
DCCP #128 SFSTART
DCCP #128 SFLENGTH
@"
```


Syntax	Example	Notes
	`$(`*x*`)`*Exp* `CON(`*x*`)`*Exp* `EXP(`*x*`)`*Exp*	Places the value of *Exp* in the code, on *x* nibbles.
	`¢`*Ascii* `"`*Ascii*`"`	Includes ASCII data. The end of the string is the next ¢ or carriage return. Example: ¢Hello¢. To output a ¢ character, put it twice. To put an char from its number, use **\xx** where xx is an hex number. To put a \, put the \ chr twice.
	`GOIN5` *lab* `G5` *lab* `GOIN4` *lab* `G4` *lab* `GOIN3` *lab* `G3` *lab* `GOIN2` *lab* `G2` *lab*	Same as `$(`*x*`)`*label*`-&` with *x*=5, 4, 3 or 2. Useful to create a jump table.
	`SAVE`	Equivalent to `GOSBVL SAVPTR`
	`LOAD`	Equivalent to `GOSBVL GETPTR`
	`RPL or LOOP`	Equivalent to `A=DAT0.A D0+5 PC=(A)`
	`LOADRPL`	Equivalent to `GOVLNG GETPTRLOOP`
	`INTOFF2`	Equivalent to `GOSBVL DisableIntr`
	`INTON2`	Equivalent to `GOSBVL AllowIntr`
	`ERROR_C`	Equivalent to `GOSBVL ErrjmpC`
	`A=IN2`	Equivalent to `GOSBVL AINRTN`
	`C=IN2`	Equivalent to `GOSBVL CINRTN`
	`OUT=C=IN`	Equivalent to `GOSBVL OUTCINRTN`
	`RES.STR`	Equivalent to `GOSBVL MAKE$N`
	`RES.ROOM`	Equivalent to `GOSBVL GETTEMP`
	`RESRAM`	Equivalent to `GOSBVL MAKERAM$`
	`SHRINK$`	Equivalent to `GOSBVL SHRINK$`
	`COPY<- COPY← COPYDN`	Equivalent to `GOSBVL MOVEDOWN`
	`COPY-> COPY→ COPYUP`	Equivalent to `GOSBVL MOVEUP`
	`DISP`	Equivalent to `GOSBVL DBUG` (only if debug is on)
	`DISPKEY`	Equivalent to `GOSBVL DBUG.KEY` (only if debug is on)
	`SRKLST`	Equivalent to `GOSBVL SHRINKLIST`
	`SCREEN`	Equivalent to `GOSBVL D0->Row1`
	`MENU`	Equivalent to `GOSBVL D0->Sft1`
	`ZEROMEM`	Equivalent to `GOSBVL WIPEOUT`
	`MULT.A`	Equivalent to `GOSBVL MULTBAC`
	`MULT`	Equivalent to `GOSBVL MPY`
	`DIV.A`	Equivalent to `GOSBVL IntDiv`
	`DIV`	Equivalent to `GOSBVL IDIV`
	`BEEP`	Equivalent to `GOSBVL makebeep`
	`NATIVE? $hex`	Set carry if native function `hex` is undefined, clear it if defined.
	`HST=1.x`	Sets bits in the HST register (XM=1, SB=1, SR=1 and MP=1 are also available). Note: the program `ST=0.0 SB=1 ?SB=0 { ST=1.0 }` will set ST0 to 0 if the calculator is non-emulated and to 1 if it is emulated.
	`?HST=1.x { }`	Test for HST bits. See HST=1.x comments
	`SETFLD(1-7)`	See the section on SETFLDn above.
	`OFF`	Turns the calculator off.

Operation	Assembler	Action	S flags
Copy and shift	MOV{cond}{S} Rd, <Oprnd>	d:= <Oprnd>	NZCR
Not	MVN{cond}{S} Rd, <Oprnd>	d:= ~<Oprnd>	NZCR
Add	ADD{cond}{S} Rd, Rn, <Oprnd>	d:= Rn + <Oprnd>	NZCVR
Add with carry	ADC{cond}{S} Rd, Rn, <Oprnd>	d:= Rn + <Oprnd> + Carry	NZCVR
Sub	SUB{cond}{S} Rd, Rn, <Oprnd>	d:= Rn - <Oprnd>	NZCVR
Sub with carry	SBC{cond}{S} Rd, Rn, <Oprnd>	d:= Rn - <Oprnd> - Not(Carry)	NZCVR
Reverse Sub	RSB{cond}{S} Rd, Rn, <Oprnd>	d:= <Oprnd> - Rn	NZCVR
Rev sub with carry	RSC{cond}{S} Rd, Rn, <Oprnd>	d:= <Oprnd> - Rn – Not(Carry)	NZCVR
Multiply	MUL{cond}{S} Rd, Rm, Rs	d:= Rm * Rs	NZR
Multiply Add	MLA{cond}{S} Rd, Rm, Rs, Rn	d:= (Rm * Rs) + Rn	NZR
Compare	CMP{cond} Rd, <Oprnd>	flags:= Rn - <Oprnd>	NZCV
Cmp Negative	CMN{cond} Rd, <Oprnd>	flags:= Rn + <Oprnd>	NZCV
Test	TST{cond} Rn, <Oprnd>	flags:= Rn And <Oprnd>	NZC
Test equivalence	TEQ{cond} Rn, <Oprnd>	flags:= Rn Xor <Oprnd>	NZC
And	AND{cond}{S} Rd, Rn, <Oprnd>	Rd:= Rn And <Oprnd>	NZC
Xor	EOR{cond}{S} Rd, Rn, <Oprnd>	Rd:= Rn Xor <Oprnd>	NZC
	XOR{cond}{S} Rd, Rn, <Oprnd>	Rd:= Rn Xor <Oprnd>	NZC
Or	ORR{cond}{S} Rd, Rn, <Oprnd>	Rd:= Rn Or <Oprnd>	NZC
BitClear (~And)	BIC{cond}{S} Rd, Rn, <Oprnd>	Rd:= Rn And Not <Oprnd>	NZC
Branch	B{cond} label	R15/PC:= address	
Gosub	BL{cond} label	R14:=R15/PC, R15/PC:= address	
Load Int	LDR{cond} Rd, <a_mode>	Rd:= [address]	
	LDR{cond} Rd, Label	Rd:= data at label. The label address is calculated relative to the PC. This does not work with constants	
Load Byte	LDR{cond}B Rd, <a_mode>	Rd:= [byte at address] 0 extended	
	LDRB{cond} Rd, Label	Rd:= data at label. The label address is calculated relative to the PC. This does not work with constants	
Multiple load		Stack operations (Pop)	
Inc Before	LDM{cond}IB Rd{!}, {reg list}	! sets the W bit (updates the base	
Inc After	LDM{cond}IA Rd{!}, {reg list}	register after the transfer)	
Dec Before	LDM{cond}DB Rd{!}, {reg list}		
Dec After	LDM{cond}DA Rd{!}, {reg list}		
Store Int	STR{cond} Rd, <a_mode>	[address]:= Rd	
	STR{cond} Rd, Label	data at label:= Rd. The label address is calculated relative to the PC. This does not work with constants	
Store Byte	STRB{cond} Rd, <a_mode>	[address]:= byte value from Rd	
	STRB{cond} Rd, Label	data at label:= Rd. The label address is calculated relative to the PC. This does not work with constants	
Multiple Store		Stack operations (Push)	
Inc Before	STM{cond}IB Rd{!}, {reg list}	! sets the W bit (updates the base	
Inc After	STM{cond}IA Rd{!}, {reg list}	register after the transfer)	
Dec Before	STM{cond}DB Rd{!}, {reg list}		
Dec After	STM{cond}DA Rd{!}, {reg list}		
Multiplication	MUL rd, r1 r2	rd=r1*r2	
	MLA rd, r1, r2, r3	rd=r1*r2+r3	
	SMULL rd1, rd2, r1, r2	Signed mul rd1=low r1*r2, rd2=high r1*r2	
	SMLAL rd1, rd2, r1, r2	Signed mul add rd1+=low r1*r2, rd2+=high r1*r2	
	UMULL rd1, rd2, r1, r2	rd1=low r1*r2, rd2=high r1*r2	
	UMLAL rd1, rd2, r1, r2	mul add rd1+=low r1*r2, rd2+=high r1*r2	
*labelName		Creates a label	
$		See $ in ASM mode	
ε, ¢		See ASM mode	

Syntax	Example	Notes
	SETOFFD	Set offset of display inside disp0 in bytes from C[X]&7FF.
	HSCREEN	Return how many lines the screen contains to Ca.
	UNCNFGD	Unconfigure the 4KB block containing the top 16-line header. This will refresh the header on the display.
	GETTIME	Emulates gettime function in ROM, and also updates the 8192Hz timer accordingly. Purpose: Get current time: (=NEXTIRQ)-Timer2 Return CS iff time appears corrupt. Entry: Timer2 Running Exit: Timer2 Running CC - A:NEXTIRQ (ticks) C:Time (ticks) D:Timer2 (sign extended ticks) P:0, HEX CS - Same as the non-error case but the time system is corrupt because of one of: (1) TIMESUM # CRC(NEXTIRQ) -- CheckSum Error (2) TIMER2 was not running on entry. (3) Time not in range: [BegofTime, EndofTime)
	MIDAPP?	Carry=1 on HP 48gII, 0 otherwise.
	CONFIGD	Configure a 4KB block containing the top 16-line header. C.A = Start address of the block (must be multiple of 4KB). If already configured, unconfig, refresh and re-config.
	BIGAPP?	Carry=1 on HP 49g+ or 50g, 0 otherwise.
	RESETOS	Reset the calculator (including the OS). This code doesn't return, the calculator restarts at 00000000.
	REFRESHD	Force to refresh the header on the display.
	AUTOTEST	- 003AA: AUTO_USER_TEST - 003A3: MANU_USER_TEST - 0039C: MANUFACTURE_TEST - Other: signed index, OS-specific, see OS_API.doc.
	ACCESSSD	SD Card functions (depending on P, see HP's vgeraccess for more details.
	PORTTAG?	Return port number depending on tag name. Entry: D1: name (size+chars) Exit: A[A]: port number (0-3) D1: after name Carry: clear if ok, set if wrong name

Messages Listed Numerically (continued)

# (hex)	Message
BB22	Y-Col:
BB23	Model:
BB24	Enter indep column number
BB25	Enter dependent column number
BB26	Choose statistical model
BB27	Correlation
BB28	Covariance
BB29	PREDICT VALUES
BB2A	Y:
BB2B	Enter indep value or press PRED
BB2C	Enter dep value or press PRED
BB2D	SUMMARY STATISTICS
BB2E	Calculate:
BB2F	ΣX
BB30	ΣY
BB31	ΣX2
BB32	ΣY2
BB33	ΣXY
BB34	NΣ
BB35	Calculate sum of X column?
BB36	Calculate sum of Y column?
BB37	Calculate sum of squares of X?
BB38	Calculate sum of squares of Y?
BB39	Calculate sum of products?
BB3A	Calculate number of data points?
BB3B	Linear Fit
BB3C	Logarithmic Fit
BB3D	Exponential Fit
BB3E	Power Fit
BB3F	Best Fit
BB40	5.Hypoth. tests…
BB41	6.Conf. interval…
	Time and Alarm Prompts
BC01	1.Browse alarms…
BC02	2.Set alarm…
BC03	3.Set time, date…
BC04	SET ALARM

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
LAST CMD Disabled	← CMD pressed while that recovery feature disabled.	125
LAST STACK Disabled	→ UNDO pressed while that recovery feature disabled.	124
LASTARG Disabled	→ ARG pressed while that recovery feature disabled.	205
Low Battery	System batteries too low to safely print or perform I/O.	C14
Many or No Solutions	A value for *I%YR* cannot be calculated. Check the values stored in *PV*, *PMT*, and *FV*. Check for correct signs.	E602
Memory Clear	Memory was cleared.	005
Name Conflict	Execution of | (where) attempted to assign value to variable of integration or summation index.	13C
Name the equation, press ENTER	Name equation and store it in *EQ*.	60B
Name the stat data, press ENTER	Name statistics data and store it in *ΣDAT*.	621
Negative Underflow	Math exception: Calculation returned negative, non-zero result greater than –MINR.	302
No Current Equation	Solver, DRAW, or RCEQ executed with nonexistent *EQ*.	104
No current equation.	Plot and HP Solve application status message.	609
No Room in Port	Insufficient free memory in the specified port.	00B
No Room to Save Stack	Not enough free memory to save copy of the stack. LAST STACK is automatically disabled.	101
No Room to Show Stack	Stack objects displayed by type only due to low memory condition.	131
No stat data to plot	No data stored in *ΣDAT*.	60F
No Solution	A value for *I%YR* cannot be calculated. Check the values stored in *PV*, *PMT*, and *FV*. Check for correct signs.	E601

Messages Listed Numerically (continued)

# (hex)	Message
BA28	Remote PC files
BA29	Files in␣
BA2A	Enter name of dir to change to
BA2B	Choose Remote Directory
BA2C	Infrared
BA2D	IR
BA2E	Wire
BA2F	Kermit
BA30	XModem
BA31	Odd
BA32	Even
BA33	Mark
BA34	Space
BA35	Spc
BA36	ASCII
BA37	ASC
BA38	Binary
BA39	Bin
BA3A	None
BA3B	Newline (Ch 10)
BA3C	Newl
BA3D	Chr 128-159
BA3E	→159
BA3F	→255
BA40	Chr 128-255
BA41	One-digit arith
BA42	Two-digit arith
BA43	Three-digit CRC
BA44	HP-IR
BA45	IrDA
BA46	14K
BA47	19K
BA48	38K
BA49	57K
BA4A	115K
BA4B	15K
BA4C	1200

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Protocol Error	Received a packet whose length was shorter than a null packet. Maximum packet length parameter from other machine is illegal.	C07
Receive Buffer Overrun	Kermit: More than 255 bytes of retries sent before calculator received another packet. SRECV: Incoming data overflowed the buffer.	C04
Receive Error	UART overrun of framing error.	C03
Receiving	Identifies object name while receiving.	C0E
Retry #	Indicates number of retries while retrying packet exchange.	C0B
Select a model	Select statistics curve fitting model.	614
Select plot type	Select plot type.	60C
Select repeat interval	Select alarm repeat interval.	61B
Sending	Identifies object name while sending.	C0D
Sign Reversal	HP Solve application or ROOT unable to find point at which current equation evaluates to zero, but did find two neighboring points at which equation changed sign.	A05
Single Equation	Only one equation supplied to Multiple-Equation Solver. Use HP Solve application.	E402
Timeout	Printing to serial port: Received XOFF and timed out waiting for XON.	C02
Too Few Arguments	Command required more arguments than were available on stack.	201
Too Many Unknowns	Multiple Equation Solver can't calculate a value given the current knowns. Supply another value or add an equation.	E404
Transfer Failed	Ten successive attempts to receive a good packet were unsuccessful.	C06
Unable to find root	PROOT is unable to determine all roots of the polynomial.	C001
Unable to Isolate	ISOL failed because specified name absent or contained in argument a function with no inverse.	130

Messages Listed Numerically (continued)

# (hex)	Message
B9BF	82 EQW edit mini fnt
B9C0	83 Display grobs off
B9C1	85 SysRPL stk disp
B9C2	90 CHOOSE:mini font
B9C3	91 MTRW:list of list
B9C4	92 MASD SysRPL mode
B9C5	94 Result <> LASTCMD
B9C6	95 Algebraic mode
B9C7	97 List:vert disp
B9C8	98 Vector:vert disp
B9C9	99 CAS:verbose
B9CA	100 Step by step on
B9CB	103 Complex on
B9CC	105 Approx. mode on
B9CD	106 !Simp. in series
B9CE	109 Num. factorize
B9CF	110 Large matrices
B9D0	111 !Simp non rat.
B9D1	112 i not simplified
B9D2	113 Linear simp off
B9D3	114 Disp x+1 → 1+x
B9D4	115 SQRT !simplified
B9D5	116 Prefer sin()
B9D6	117 Soft MENU
B9D7	119 Rigorous off
B9D8	120 Silent mode on
B9D9	123 Forb. Switch Mode
B9DA	125 FastSign-no Sturm
B9DB	126 rref w/o last col
B9DC	127 HP-IR mode
B9DD	128 Vars are reals
	I/O Prompts
B9DE	Object:
B9DF	Obs in
B9E0	Name:
BA01	1.Send to Calculator…
BA02	2.Get from Calculator

Messages Listed Numerically

# (hex)	Message
	General Messages
001	Insufficient Memory
002	Directory Recursion
003	Undefined Local Name
004	Undefined XLIB Name
005	Memory Clear
006	Power Lost
007	Warning:
008	Invalid Card Data
009	Object In use
00A	Port Not Available
00B	No Room in Port
00C	Object Not in Port
00D	Recovering Memory
00E	Try To Recover Memory?
00F	Replace RAM, Press ON
010	No Mem To Config All
011	Undefined FPTR Name
012	Invalid Bank Data
013	Full Check Bad Crc
014	Cmprs: not a user bank
015	No or 2 system bank
016	Invalid bank
017	Invalid bank number
018	Inexisting pack
019	Pack twice
01A	Ins. Mem.␣
01B	Erase Fail, Rom faulty
01C	Erase Fail, Low bats
01D	Erase Fail, Locked Block
01E	Write Adr outside ROM
01F	Write Fail, Rom Faulty
020	Write Fail, Low bats
021	Write Fail, Locked Block
022	Invalid DOS Name
023	File already opened
024	Invalid File Handle
025	Invalid File Index

Messages Listed Numerically (continued)

# (hex)	Message
B976	100 Step by step off
B977	103 Complex off
B978	105 Exact mode on
B979	106 Simp. in series
B97A	109 Sym. factorize
B97B	110 Normal matrices
B97C	111 Simp non rat.
B97D	112 i simplified
B97E	113 Linear simp on
B97F	114 Disp 1+x → x+1
B980	115 SQRT simplified
B981	116 Prefer cos()
B982	117 CHOOSE boxes
B983	119 Rigorous on
B984	120 Silent mode off
B985	123 Allow Switch Mode
B986	125 Accur. Sign-Sturm
B987	126 rref w/ last col
B988	127 IrDA mode
B989	128 Cmplx var allowed
B98A	01 Principal value
B98B	02 Constant → num
B98C	03 Function → num
B98D	14 Payment at begin
B98E	19 V2 → complex
B98F	20 Underflow → error
B990	21 Overflow → error
B991	22 Infinite → ±9E499
B992	27 'X+Y*i' → 'X+Y*i'
B993	28 Simultaneous plot
B994	29 Don't draw axes
B995	31 Plot points only
B996	32 Inverse cursor
B997	33 Transfer via IR
B998	34 Print via wire
B999	35 Binary transfer
B99A	36 RECV overwrites

Messages Listed Numerically (continued)

# (hex)	Message
11D	Long Complex
11E	Linked Array
11F	Character
120	Code
121	Library Data
122	External
123	(#123h DOERR is equivalent to KILL)
124	LAST STACK Disabled
125	LAST CMD Disabled
126	HALT Not Allowed
127	Array
128	Wrong Argument Count
129	Circular Reference
12A	Directory Not Allowed
12B	Non-Empty Directory
12C	Invalid Definition
12D	Missing Library
12E	Invalid PPAR
12F	Non-Real Result
130	Unable to Isolate
131	No Room to Show Stack
132	Warning:
133	Error:
	Out-of-Memory Prompts
134	Purge?
135	Out of Memory
136	Stack
137	Last Stack
138	Last Commands
139	Key Assignments
13A	Alarms
13B	Last Arguments
13C	Name Conflict
13D	Command Line
13E	(#13Eh DOERR is equivalent to CONT)
13F	Interrupted
140	Integer

# (hex)	Message
B92C	Degrees
B92D	Deg
B92E	Radians
B92F	Rad
B930	Grads
B931	Grad
B932	Rectangular
B933	Polar
B934	Spherical
	System Flags Choose Box Prompts
B935	SYSTEM FLAGS
B936	01 General solutions
B937	02 Constant → symb
B938	03 Function → symb
B939	14 Payment at end
B93A	19 →V2 → vector
B93B	20 Underflow → 0
B93C	21 Overflow → ±9E499
B93D	22 Infinite → error
B93E	27 'X+Y*i' → '(X,Y)'
B93F	28 Sequential plot
B940	29 Draw axes too
B941	31 Connect points
B942	32 Solid cursor
B943	33 Transfer via wire
B944	34 Print via IR
B945	35 ASCII transfer
B946	36 RECV renames
B947	37 Single-space prnt
B948	38 Add linefeeds
B949	39 Show I/O messages
B94A	40 Don't show clock
B94B	41 12-hour clock
B94C	42 mm/dd/yy format
B94D	43 Reschedule alarm
B94E	44 Delete alarm
B94F	51 Fraction mark: .
B950	52 Show many lines

# (hex)	Message
	Floating-Point Errors
301	Positive Underflow
302	Negative Underflow
303	Overflow
304	Undefined Result
305	Infinite Result
	Array Messages
501	Invalid Dimension
502	Invalid Array Element
503	Deleting Row
504	Deleting Column
505	Inserting Row
506	Inserting Column
	Statistics Messages
601	Invalid Σ Data
602	Nonexistent ΣDAT
603	Insufficient Σ Data
604	Invalid ΣPAR
605	Invalid Σ Data LN(Neg)
606	Invalid Σ Data LN(0)
	Plot, I/O, Time and HP Solve Application Messages
607	Invalid EQ
608	Current equation:
609	No current equation.
60A	Enter eqn, press NEW
60B	Name the equation, press ENTER
60C	Select plot type
60D	Empty catalog
60E	undefined
60F	No stat data to plot
610	Autoscaling
611	Solving for_
612	No current data. Enter
613	data point, press Σ+
614	Select a model
615	No alarms pending.
616	Press ALRM to create
617	Next alarm:
618	Past due alarm:
619	Acknowledged

# (hex)	Message
	Unit Management
B01	Invalid Unit
B02	Inconsistent Units
	I/O and Printing
C01	Bad Packet Block check
C02	Timeout
C03	Receive Error
C04	Receive Buffer Overrun
C05	Parity Error
C06	Transfer Failed
C07	Protocol Error
C08	Invalid Server Cmd.
C09	Port Closed
C0A	Connecting
C0B	Retry #
C0C	Awaiting Server Cmd.
C0D	Sending␣
C0E	Receiving␣
C0F	Object Discarded
C10	Packet #
C11	Processing Command
C12	Invalid IOPAR
C13	Invalid PRTPAR
C14	Low Battery
C15	Empty Stack
C16	Row␣
C17	Invalid Name
	Time Messages
D01	Invalid Date
D02	Invalid Time
D03	Invalid Repeat
D04	Nonexistent Alarm
	A Programmer's DOERR
DFF	(Causes silent interruption, leaving all pending HALTs intact. This is the action performed before control alarms are executed)
	Input Form Prompts
B901	Press [CONT] for menu
B902	reset/delete this field
B903	Reset value
B904	Delete value

# (hex)	Message
70F	Polar
710	Spherical
711	Operating Mode…
712	Number Format……
713	Angle Measure……
714	Coord System……
715	FM;
716	Beep
717	Key Click
718	Last Stack
719	Choose calculator operating mode
71A	Choose number display format
71B	Choose decimal places to display
71C	Choose angle measure
71D	Choose coordinate system
71E	Use comma as fraction mark?
71F	Enable standard beep?
720	Enable key click?
721	Save last stk for UNDO and ANS?
722	CALCULATOR MODES
723	Font:
724	Stack:
725	Small
726	Textbook
727	Edit:
728	Small
729	Full Page
72A	Indent
72B	EQW:
72C	Small
72D	Small Stack Disp
72E	Header:
72F	Clock
730	Analog
731	Choose system font
732	Display stack using small font?
733	Use pretty print in the stack?
734	Edit using small font?

# (hex)	Message
876	Confidence level
877	CONF. INT.: 2 P
878	x̄:
879	Sx:
87A	N:
87B	C:
87C	Sample mean
87D	Sample standard deviation
87E	Sample size
87F	Confidence level
880	CONF. INT.: 1 μ, UNKNOWN σ
881	x̄1:
882	S1:
883	N1:
884	C:
885	x̄2:
886	S2:
887	N2:
888	Pooled
889	Sample 1 mean
88A	Std deviation for sample 1
88B	Sample 1 size
88C	Sample 2 mean
88D	Std deviation for sample 2
88E	Sample 2 size
88F	Confidence level
890	Pooled if checked
891	CONF. INT.: 2 μ, UNKNOWN σ
	Object Editing Messages
892	Search for:
893	Replace by:
894	Case Sensitive
895	Search For:
896	Enter search pattern
897	Enter replace pattern
898	Case sensitive search?
899	Enter search pattern

# (hex)	Message
75B	Zoom:
75C	Small Font
75D	File:
75E	Enter starting value
75F	Enter increment value
760	Choose table format
761	Enter zoom factor
762	Display table using small font?
763	Enter a filename to save data
764	TABLE SETUP
765	Automatic
766	Build Your Own
767	Function
768	Polar
769	Parametric
76A	Diff Eq
76B	Conic
76C	Truth
76D	Histogram
76E	Bar
76F	Scatter
770	Slopefield
771	Fast3D
772	Wireframe
773	Ps-Contour
774	Y-Slice
775	Gridmap
776	Pr-Surface
777	Deg
778	Rad
779	Grad
77A	Type:
77B	∠:
77C	EQ:
77D	Indep:
77E	Connect
77F	Simult
780	H-Tick:

Messages Listed Numerically (continued)

# (hex)	Message
82A	X2:
82B	N2:
82C	Success count for sample 1
82D	Size of sample 1
82E	Significance level
82F	Success count for sample 2
830	Size of sample 2
831	Z-TEST: 2 P
832	$\bar{x}$:
833	Sx:
834	μ0:
835	α:
836	N:
837	Null hypothesis population mean
838	Sample Standard deviation
839	Sample Mean
83A	Significance level
83B	Sample size
83C	T-TEST: 1 μ, UNKNOWN σ
83D	$\bar{x}$1:
83E	S1:
83F	N1:
840	α:
841	$\bar{x}$2:
842	S2:
843	N2:
844	Pooled?
845	Sample mean for population 1
846	Std deviation for sample 1
847	Sample size for population 1
848	Significance level
849	Sample mean for population2
84A	Std deviation for sample 2
84B	Sample size for population 2
84C	"Pooled" if checked
84D	T-TEST: 2 μ, UNKNOWN σ
84E	$\bar{x}$:
84F	σ:

Messages Listed Numerically (continued)

# (hex)	Message
7A8	Depnd Low:
7A9	High:
7AA	X-Left:
7AB	X-Right:
7AC	Y-Near:
7AD	Y-Far:
7AE	Step Indep:
7AF	Depnd:
7B0	Bar Width:
7B1	Z-Low:
7B2	Z-High:
7B3	XE:
7B4	YE:
7B5	ZE:
7B6	Init:
7B7	Final:
7B8	Init-Soln:
7B9	Tol:
7BA	XXLeft:
7BB	XXRight:
7BC	YYNear:
7BD	YYFar:
7BE	Enter minimum horizontal value
7BF	Enter maximum horizontal value
7C0	Enter minimum vertical value
7C1	Enter maximum vertical value
7C2	Enter minimum indep var value
7C3	Enter maximum indep var value
7C4	Enter indep var increment
7C5	Indep step units are pixels?
7C6	Enter minimum depend var value
7C7	Enter maximum depend var value
7C8	Enter bar width
7C9	Enter minimum Z view-volume val
7CA	Enter maximum Z view-volume val
7CB	Enter X eyepoint coordinate
7CC	Enter Y eyepoint coordinate

Messages Listed Numerically (continued)

# (hex)	Message
BC05	Message:
BC06	Time:
BC07	Date:
BC08	Repeat:
BC09	Enter "message" or « action »
BC0A	Enter hour
BC0B	Enter minute
BC0C	Enter second
BC0D	Choose AM, PM, or 24-hour time
BC0E	Enter month
BC0F	Enter day
BC10	Enter year
BC11	Enter alarm repeat multiple
BC12	Enter alarm repeat unit
BC13	SET TIME AND DATE
BC14	Choose date display format
BC15	Monday
BC16	Tuesday
BC17	Wednesday
BC18	Thursday
BC19	Friday
BC1A	Saturday
BC1B	Sunday
BC1C	None
BC1D	AM
BC1E	PM
BC1F	24-hour time
BC20	24-hr
BC21	1 January
BC22	2 February
BC23	3 March
BC24	4 April
BC25	5 May
BC26	6 June
BC27	7 July
BC28	8 August
BC29	9 September

System Flags (continued)

Flag	Description
–112	Simplifying 'i'. *Clear*: 'i' can be simplified (i.e. $i^2 = -1$) *Set*: 'i' cannot be simplified.
–113	Linear Simplification Mode. *Clear*: Apply linearity simplification when using integration CAS commands. *Set*: Do not apply linearity simplification when using integration CAS commands.
–114	Polynomial Term Order. *Clear*: Polynomial expressed in decreasing power order. *Set*: Polynomial expressed in increasing power order.
–115	SQRT Simplification. *Clear*: Square roots can be simplified. *Set*: Square roots cannot be simplified.
–116	Trigonometric Manipulations. *Clear*: Simplification to cosine terms. *Set*: Simplification to sine terms.
–117	Menu Display Mode. *Clear*: Menus displayed as choose boxes. *Set*: Menus displayed as softkeys, like the HP 48GX calculator.
–118	INT Simplification. *Clear*: INT is simplified. *Set*: INT is not simplified.
–119	Rigorous Mode. *Clear*: Rigorous mode on: |X| is not simplified to X. *Set*: Rigorous mode off: |X| is simplified to X.
–120	Silent Mode Switch. *Clear*: Calculator prompts when it needs to change modes. *Set*: Calculator changes modes when necessary without prompting.
–121	*Internal use only.* (LN returns LN[ABS()] if set).
–122	*Internal use only.* (0/0 occurred).
–123	Mode Switch. *Clear*: Mode switch allowed. *Set*: Mode switch not allowed.
–124	CAS Object Evaluation. *Clear*: Non-algebraic CASCOMPEVAL is allowed. *Set*: Non-algebraic CASCOMPEVAL is not allowed.
–125	Sign Determination Mode. *Clear*: Accurate sign determination using polynomial Sturm sequences. *Set*: Fast sign determination. Polynomial Sturm sequences are not used. Auto-simplification of square roots canceled.
–126	Row Reduction Mode. *Clear*: RREF done with last column. *Set*: RREF done without last column.
–127	*Not used.*
–128	*Clear*: Complex variables allowed. *Set* (default): All variables are real.

Messages Listed Numerically (continued)

# (hex)	Message
BD13	Enter variable value
BD14	Expression
BD15	TAYLOR POLYNOMIAL
BD16	Order:
BD17	Enter Taylor polynomial order
BD18	ISOLATE A VARIABLE
BD19	Principal
BD1A	Get principal solution only?
BD1B	SOLVE QUADRATIC
BD1C	MANIPULATE EXPRESSION
BD1D	MATCH EXPRESSION
BD1E	Pattern:
BD1F	Replacement:
BD20	Subexpr First
BD21	Cond:
BD22	Enter pattern to search for
BD23	Enter replacement object
BD24	Search subexpressions first?
BD25	Enter conditional expression
BD26	Symbolic
BD27	Numeric
	Plot Application Prompts
BE01	Plot
BE02	Type:
BE03	∠:
BE04	H-View:
BE05	Autoscale
BE06	V-View:
BE07	Choose type of plot
BE08	Choose angle measure
BE09	Enter function(s) to plot
BE0A	Enter minimum horizontal value
BE0B	Enter maximum horizontal value
BE0C	Autoscale vertical plot range?
BE0D	Enter minimum vertical value
BE0E	Enter maximum vertical value
BE0F	Plot (x(t), y(t))

System Flags (continued)

Flag	Description
–79	Pretty Print Mode. *Clear*: Algebraic objects appear on the stack in textbook (EQW) form. (Only in multi-line levels, see flag –65). *Set*: Algebraic objects appear on the stack in linear form.
–80	Font used to show algebraics on stack if flag –79 is clear. *Clear*: Textbook stack display uses the current system font. *Set*: Textbook stack display uses mini-font.
–81	Font used by →GROB on algebraics. *Clear*: Editing a textbook grob uses current font. *Set*: Editing a textbook grob uses mini-font.
–82	Equation Writer Font. *Clear*: Current font used to edit algebraics in textbook mode. *Set*: Mini-font used to edit algebraics in textbook mode.
–83	Grob Display. *Clear*: Grob contents (picture) displayed on the stack. *Set*: Grob description (dimensions) displayed on the stack.
–84	*Not used.* (Originally intended to control the menu font size in the HP 49G, though it was never implemented.)
–85	Stack Display. *Clear*: Standard stack display. *Set*: System-RPL stack display. In textbook mode (see flag –79), objects displayed on multiple lines (see flag –65) are always shown in standard form.
–86	Program Prefix. *Clear*: Program prefix off. *Set*: Program prefix on.
–87	Recursive Stack Display. *Clear*: Non-recursive stack display. *Set*: In System-RPL stack display (see flag –85), unsupported (unnamed) entry points are exploded into their elements.
–88	*Not used.* (Originally intended to control recursive editing in the 49G, though it was never implemented.)
–89	*Not used.* (Originally intended to control extable library usage editing in the HP 49G, though it was never implemented.)
–90	Choose Box Font. *Clear*: Choose boxes displayed in current font. *Set* (default): Choose boxes displayed in mini-font.
–91	Matrix Writer Object Type. *Clear*: Matrix Writer returns arrays only, like the HP 48GX calculator. *Set*: Matrix Writer returns a list of lists.
–92	Assembler Mode. *Clear*: Assembler defaults to making code objects. *Set*: Assembler defaults to making System-RPL programs.
–93	Erable Header. *Not used.*

Messages Listed Numerically (continued)

# (hex)	Message
BE35	Tick spacing units are pixels?
BE36	Depnd:
BE37	Enter dependent var name
BE38	Enter minimum dep var value
BE39	Enter maximum dep var value
BE3A	H-Var:
BE3B	V-Var:
BE3C	Enter max indep var increment
BE3D	Choose horizontal variable
BE3E	Choose vertical variable
BE3F	0 INDEP
BE40	1 SOLN
BE41	SOLN(
BE42	X-Left:
BE43	X-Right:
BE44	Y-Near:
BE45	Y-Far:
BE46	Z-Low:
BE47	Z-High:
BE48	Enter minimum X view-volume val
BE49	Enter maximum X view-volume val
BE4A	Enter minimum Y view-volume val
BE4B	Enter maximum Y view-volume val
BE4C	Enter minimum Z view-volume val
BE4D	Enter maximum Z view-volume val
BE4E	XE:
BE4F	YE:
BE50	ZE:
BE51	Enter X eyepoint coordinate
BE52	Enter Y eyepoint coordinate
BE53	Enter Z eyepoint coordinate
BE54	Save Animation
BE55	Save animation data after plot?
BE56	XX-Left:
BE57	XX-Rght:
BE58	YY-Near:
BE59	YY-Far:

System Flags (continued)

Flag	Description
–53	Precedence. *Clear*: Certain parentheses in algebraic expressions suppressed to improve legibility. *Set*: All parentheses in algebraic expressions displayed.
–54	Tiny Array Elements. *Clear*: Singular values computed by RANK (and other commands that compute the rank of a matrix) that are more than 1×10^{-14} times smaller than the largest computed singular value in the matrix are converted to zero. Automatic rounding for DET is enabled. *Set*: Small computed singular values (see above) not converted. Automatic rounding for DET is disabled.
–55	Last Arguments. *Clear*: Command arguments saved. *Set*: Command arguments not saved.
–56	Error Beep. *Clear*: Error, key click and BEEP-command beeps enabled. *Set*: Error, key click and BEEP-command beeps suppressed.
–57	Alarm Beep. *Clear*: Alarm beep enabled. *Set*: Alarm beep suppressed.
–58	Verbose Messages. *Clear*: Parameter variable data automatically displayed. *Set*: Automatic display of parameter variable data is suppressed.
–59	*No longer used.* (It was the Fast Catalog/Browser Display flag in the HP 48SX/GX).
–60	Alpha Lock. *Clear*: Single-Alpha activated by pressing ALPHA once. Alpha lock activated by pressing ALPHA twice. *Set*: Alpha lock activated by pressing ALPHA once. (Single-Alpha not available.)
–61	User-Mode Lock. *Clear*: 1-User mode activated by pressing ← USER once. User mode activated by pressing ← USER twice. *Set*: User mode activated by pressing ← USER once. (1-User mode not available.)
–62	User Mode. *Clear*: User mode not active. *Set*: User mode active.
–63	Vectored ENTER. *Clear*: ENTER evaluates command line. *Set*: User-defined ENTER activated.
–64	Index Wrap Indicator. *Clear*: Last execution of GETI or PUTI did not increment index to first element. *Set*: Last execution of GETI or PUTI did increment index to first element.

Messages Listed Numerically (continued)

# (hex)	Message
BF07	Enter value or press SOLVE
BF08	Eq:
BF09	Enter function to solve
BF0A	Funcs in
BF0B	Solver Variable Order
BF0C	Variables:
BF0D	Enter order of vars to display
BF0E	SOLVE Y'(T)=F(T,Y)
BF0F	f:
BF10	∂f∂y:
BF11	∂f∂t:
BF12	Indep:
BF13	Init:
BF14	Final:
BF15	Soln:
BF16	Tol:
BF17	Step:
BF18	Stiff
BF19	Enter function of INDEP and SOLN
BF1A	Enter derivative w.r.t. SOLN
BF1B	Enter derivative w.r.t. INDEP
BF1C	Enter independent var name
BF1D	Enter initial indep var value
BF1E	Enter final indep var value
BF1F	Enter solution var name
BF20	Enter initial solution var value
BF21	Press SOLVE for final soln value
BF22	Enter absolute error tolerance
BF23	Enter initial step size
BF24	Calculate stiff differential?
BF25	f
BF26	Tolerance
BF27	Solution
BF28	SOLVE AN·X^N+…+A1·X+A0
BF29	Coefficients [an … a1 a0]:
BF2A	Roots:
BF2B	Enter coefficients or press SOLVE

System Flags (continued)

Flag	Description
–20	Underflow Exception. *Clear*: Underflow exception returns 0, sets flag –23 or –24. *Set*: Underflow exception treated as an error.
–21	Overflow Exception. *Clear*: Overflow exception returns ± 9.99999999999E499 and sets flag –25. *Set*: Overflow exception treated as an error.
–22	Infinite Result Exception. *Clear*: Infinite result exception treated as an error. *Set*: Infinite result exception returns ± 9.99999999999E499 and sets flag –26.
–23 –24 –25 –26	Negative Underflow Indicator. Positive Underflow Indicator. Overflow Indicator. Infinite Result Indicator. When an exception occurs, corresponding flag (–23 through –26) is set only if the exception is not treated as an error.
–27	Display of symbolic complex numbers. *Clear*: Displays symbolic complex numbers in coordinate form (i.e. `'(x,y)'`). *Set* (default): Displays symbolic complex numbers using `'i'` (i.e. `'x+y*i'`).
–28	Simultaneous Plotting of Multiple Functions. *Clear*: Multiple equations are plotted serially. *Set*: Multiple equations are plotted simultaneously.
–29	Draw Axes. *Clear*: Axes are drawn for two-dimensional and statistical plots. *Set*: Axes are not drawn for two-dimensional and statistical plots.
–30	*Not used.*
–31	Curve Filling. *Clear*: Curve filling between plotted points enabled. *Set*: Curve filling between plotted points suppressed.
–32	Graphics Cursor. *Clear*: Graphics cursor always dark. *Set*: Graphics cursor dark on light background and light on dark background.
–33	I/O Device. *Clear*: I/O directed to USB/serial port. *Set*: I/O directed to IrDA port.
–34	Printing Device. *Clear*: Prints via IR to the HP 82240 printer. Flag –33 is ignored. *Set* (default): Printer output directed to USB/serial port if flag –33 is clear, or to IrDA compatible printer otherwise.
–35	Kermit I/O Data Format. *Clear*: Objects transmitted in ASCII form. *Set*: Objects transmitted in binary (memory image) form.

Messages Listed Numerically (continued)

# (hex)	Message
BF51	Interest:
BF52	Balance:
BF53	Enter no. of payments to amort
BF54	Principal
BF55	Interest
BF56	Balance
C001	Unable to find root
	CAS Messages
DE01	denominator(s)␣
DE02	root(s)␣
DE03	last␣
DE04	obvious␣
DE05	factorizing␣
DE06	value␣
DE07	test(s)␣
DE08	searching␣
DE09	TAYLR of ↓ at␣
DE0A	nth␣
DE0B	is␣
DE0C	numerator(s)␣
DE0D	Less than␣
DE0E	multiplicity␣
DE0F	list of␣
DE10	at␣
DE11	factor(s)␣
DE12	Eigenvalues␣
DE13	Computing for
DE14	Root mult <
DE15	Numerical to symbolic
DE16	Invalid operator
DE17	Result:
DE18	Pivots
DE19	Press CONT to go on
DE1A	Test␣
DE1B	To be implemented
DE1C	Unable to factor
DE1D	Z is not = 1 mod 4

Properties of Elements

Property	Type of Object *	Property Number
Atomic Number	Real	1
Mass Number †	Real	2
Atomic Weight	Unit	3
Density †	Unit	4
Oxidation States †	String	5
Electron Configuration	String	6
State	String	7
Melting Point	Unit	8
Boiling Point	Unit	9
Heat of Vaporization	Unit	10
Heat of Fusion	Unit	11
Specific Heat	Unit	12
Group (U.S. Customary)	String	13
Family	String	14
Crystal Structure	String	15
Atomic Volume †	Unit	16
Atomic Radius	Unit	17
Covalent Radius	Unit	18
Thermal Conductivity †	Unit	19
Electrical Conductivity †	Unit	20
First Ionization Potential	Unit	21
Electronegativity (Pauling's number)	Unit	22
Oxide Behavior	String	23
Element Name ‡	String	24
Element Symbol ‡	Name	25

* Properties returning unit objects return real objects if units aren't used. Unknown values are returned as the string "-".

† See the notes that follow this table.

‡ Not included in catalog of properties, but listed in the title of the catalog.

Notes about properties: Mass number for a stable element is based on the isotope with the highest percent abundance; for a radioactive element, it's based on the longest half-life. Density for a gas is at 273 K with units of g/l; for others, it's at 300 K with units of g/cm^3. Oxidation states are in order of most stable to least stable. Atomic volume for a gas is for its liquid state at the boiling point; for others, it's dervied from the density at 300 K. Thermal conductivity is measured at 300 K. Electrical conductivity is measured at 293 K.

Data in the Periodic Table application is based on the "Periodic Table of the Elements" published by Sargent-Welch Scientific Company, a VWR Company, and is used by permission.

Messages Listed Numerically (continued)

# (hex)	Message
DE43	Numeric input
DE44	Singularity! Continue?
DE45	Cancelled
DE46	Negative integer
DE47	Parameter is cur. var. dependent
DE48	Unsimplified sqrt
DE49	Non polynomial system
DE4A	Unable to solve ODE
DE4B	Array dimension too large
DE4C	Unable to reduce system
DE4D	Complex number not allowed
DE4E	Polyn. valuation must be 0
DE4F	Mode switch not allowed here
DE50	Non algebraic in expression
DE51	Purge current variable
DE52	Reduction result
DE53	Matrix not diagonalizable
DE54	Int[u'*F(u)] with u=
DE55	Int. by part u'*v, u=
DE56	Square root
DE57	Rational fraction
DE58	Linearizing
DE59	Risch alg. of tower
DE5A	Trig. fraction, u=
DE5B	Unknown operator (DOMAIN)
DE5C	Same points
DE5D	Unsigned inf. Solve?
DE5E	CAS not available
DE5F	Can not store current var
DE60	Not available on the HP40G
DE61	Not available on the HP49G
DE62	SERIES remainder is O(1) at order 3
DE63	Delta/Heaviside not available fromHOME
DE64	Warning, integrating in approx mode
DE65	Function is constant
DE66	Can not unbind local vars

Units (continued)

Unit (Full Name)	Value in SI Units
rd (rod)	5.02921005842 m
rem (rem)	.01 m^2 / s^2
rpm (revolutions per minute)	60 1 / s
s (second)	1 s
S (siemens)	1 $A^2 \cdot s^3$ / $kg \cdot m^2$
sb (stilb)	10000 cd / m^2
slug (slug)	14.5939029372 kg
sr (steradian)	1 sr
st (stere)	1 m^3
St (stokes)	.0001 m^2 / s
Sv (sievert)	1 m^2 / s^2
t (metric ton)	1000 kg
T (tesla)	1 kg / $A \cdot s^2$
tbsp (tablespoon)	1.47867647813 x 10^{-5} m^3
therm (EEC therm)	105506000 $kg \cdot m^2$ / s^2
ton (short ton)	907.18474 kg
tonUk (long ton (UK))	1016.0469088 kg
torr (torr (mmHg))	133.322368421 kg / $m \cdot s^2$
tsp (teaspoon)	4.92892159375 x 10^{-6} m^3
u (unified atomic mass)	1.6605402 x 10^{-27} kg
V (volt)	1 $kg \cdot m^2$ / $A \cdot s^3$
W (watt)	1 $kg \cdot m$ / s^3
Wb (weber)	1 $kg \cdot m^2$ / $A \cdot s^2$
yd (international yard)	.9144 m
yr (year)	31556925.9747 s
? (undefined)	undefined unit

Messages Listed Numerically (continued)

# (hex)	Message
DF23	RECV
DF24	HALT
DF25	VIEW
DF26	EDITB
DF27	HEADER
DF28	LIST
DF29	SORT
DF2A	XSEND
DF2B	CHDIR
DF2C	CANCL
DF2D	OK
DF2E	CHECK
DF2F	WARNING: Formatting will erase the SD card
DF30	Do you want to continue?
DF31	FORMAT
DF32	Please Wait...
	Constants Library Messages
E101	Avogadro's number
E102	Boltzmann
E103	molar volume
E104	universal gas
E105	std temperature
E106	std pressure
E107	Stefan-Boltzmann
E108	speed of light
E109	permittivity
E10A	permeability
E10B	accel of gravity
E10C	gravitation
E10D	Planck's
E10E	Dirac's
E10F	electronic charge
E110	electron mass
E111	q/me ratio
E112	proton mass
E113	mp/me ratio
E114	fine structure
E115	mag flux quantum
E116	Faraday

Units (continued)

Unit (Full Name)	Value in SI Units
°**F** (degrees Fahrenheit)	0.555555555556 K or 255.927777778 K
fath (fathom)	1.82880365761 m
fbm (board foot)	.002359737216 m^3
fc (footcandle)	10.7639104167 cd•sr / m^2
Fdy (faraday)	96487 A•s
fermi (fermi)	1 x 10^{-15} m
flam (footlambert)	3.42625909964 cd / m^2
ft (international foot)	.3048 m
ftUS (survey foot))	.304800609601 m
g (gram)	.001 kg
ga (standard freefall)	9.80665 m / s^2
gal (US gallon)	.003785411784 m^3
galC (Canadian gallon)	.00454609 m^3
galUK (UK gallon)	.004546092 m^3
gf (gram-force)	.00980665 kg•m / s^2
gmol (gram-mole)	1 mol
grad (gradian)	1.57079632679 x 10^{-2} r
grain (grain)	.00006479891 kg
Gy (gray)	1 m^2 / s^2
H (henry)	1 kg•m^2 / A^2•s^2
h (Hour)	3600 s
hp (horsepower)	745.699871582 kg•m^2 / s^3
Hz (hertz)	1 / s
in (inch)	.0254 m
inHg (inches of mercury, 0 °C)	3386.38815789 kg / m•s^2
inH2O (inches of water, 60 °F)	248.84 kg / m•s^2
J (joule)	1 kg•m^2 / s^2
K (kelvins)	1 K
kg (kilogram)	1 kg
kip (kilopound-force)	4448.22161526 kg•m / s^2
knot (nautical miles per hour)	.514444444444 m / s
kph (kilometers per hour)	.277777777778 m / s
l (liter)	.001 m^3
lam (lambert)	3183.09886184 cd / m^2
lb (avoirdupois pound)	.45359237 kg
lbf (pound -force)	4.44822161526 kg•m / s^2

Messages Listed Numerically (continued)

# (hex)	Message
E408	Searching
	Periodic Table Messages
E501	Bad Molecular Formula
E502	Undefined Element
E503	Undefined Property
	Financial Solver Messages
E601	No Solution
E602	Many or No Solutions
E603	I%YR/PYR ≤ -100
E604	Invalid N
E605	Invalid PYR
E606	Invalid #Periods
E607	Undefined TVM Variable
E608	END mode
E609	BEGIN mode
E60A	payments/year
E60B	Principal
E60C	Interest
E60D	Balance
	Development Library and Miscellaneous Messages
10001	Invalid $ROMID
10002	Invalid $TITLE
10003	Invalid $MESSAGE
10004	Invalid $VISIBLE
10005	Invalid $HIDDEN
10006	Invalid $EXTPRG
10101	Invalid File
10102	Too Many
10103	Unknown Instruction
10104	Invalid Field
10105	Val betw 0-15 expected
10106	Val betw 1-16 expected
10107	Label Expected
10108	Hexa Expected
10109	Decimal Expected
1010A	Can't Find
1010B	Label already defined
1010C	{ expected
1010D	} expected
1010E	(expected

```
*104 old soft-menu I/O (105 MENU I/O: "INPUT/OUTPUT")
*105 old soft-menu I/O SRVR (104 MENU SRVR: "SERVER")
*106 old soft-menu I/O IOPAR (no choose-box version available)
*107 old soft-menu I/O PRINT (104 MENU PRINT: "PRINT")
*108 old soft-menu I/O PRINT PRTPAR (no choose-box version available)
*109 old soft-menu I/O SERIAL (104.02 MENU SERIAL: "SERIAL IO")
*110 LIBRARY commands (110 DUP MENU MENU LS&PREV: "LIBRARY")
 111 same result as LIB (no choose-box version available)
 112 same result as LIB (no choose-box version available)
 113 APPS 12 ("EQN LIBRARY")
 114 APPS 12 EQLIB (APPS 12 1: "EQN LIB")
 115 APPS 12 COLIB (APPS 12 2: "CON LIB")
 116 APPS 12 MES (APPS 12 3: "MES LIB")
 117 APPS 12 UTILS (APPS 12 4: "UTIL LIB")
```

Menus 118 through 177

These menus were first introduced in the HP 49G calculator.

```
*118 abandoned UNITS TOOLS (see menu #59; "TOOLS")
 119 APPS CAS (APPS 11: "CAS")
 120 S.SLV (S.SLV: "S.SLV")
 121 EXP&LN (EXP&LN: "EXP&LN")
 122 TRIG (TRIG: "TRIG")
 123 CALC (CALC: "CALC")
 124 ALG (ALG: "ALG")
 125 ARITH (ARITH: "ARITH")
 126 ARITH POLY (ARITH 2: "POLYNOMIAL")
 127 ARITH INTEG (ARITH 1: "INTEGER")
 128 ARITH MODUL (ARITH 3: "MODULAR")
 129 MATRICES (MATRICES: "MATRICES")
 130 CMPLX (CMPLX: "COMPLEX")
 131 CONVERT (CONVERT: "CONVERT")
 132 RS&NUM.SLV (NUM.SLV: "NUM.SLV")
*133 soft-menu TVM (133 DUP MENU MENU LS&PREV: "FINANCE")
 134 SYMB ARITH (SYMB 2: "SYMBOLIC ARITH")
*135 abandoned SYMB CONV ("SYMBOLIC CONV")
*136 abandoned SYMB DIFF or SYMB CALC ("SYMBOLIC CALC")
*137 abandoned SYMB MATRX ("SYMBOLIC MAT")
*138 abandoned SYMB MOD ("SYMBOLIC MOD")
 139 SYMB TRIG (SYMB 6: "SYMBOLIC TRIG")
 140 CONVERT TRIG (CONVERT 3: "TRIG CONVERT")
*141 abandoned SYMB UNARY ("SYMBOLIC UNARY")
*142 abandoned SYMB BASE (meaning basic, not binary; "SYMBOLIC BASE")
 143 SYMB (SYMB: "SYMBOLIC")
*144 abandoned PRG (the MODES menu is missing; "PROG")
*145 abandoned PRG BRCH (like PRG BRCH but flat; "BRANCH")
 146 MATRICES CREAT (MATRICES 1: "MATRIX CREATE")
*147 abandoned MATRICES NORM menu (a subset of MATRICES OPER; "MATRIX NORM")
 148 MATRICES FACT (MATRICES 3: "MATRIX FACTOR.")
*149 abandoned MATRICES COL ("CREATE COL")
*150 abandoned MATRICES ROW ("CREATE ROW")
 151 SYMB ALG (SYMB 1: "SYMBOLIC ALGEBRA")
 152 SYMB CALC (SYMB 3: "SYMBOLIC CALC")
 153 SYMB GRAPH (SYMB 4: "SYMBOLIC GRAPH")
```


Reserved Variables

The calculator uses the following *reserved variables*. These have specific purposes, and their names are used as implicit arguments for certain commands. Avoid using these variables' names for other purposes, or you may interfere with the execution of the commands that use these variables.

You can change some of the values in these variables with programmable commands, while others require you to store new values into the appropriate place.

System Reserved Variables

Most system reserved variables (except *ALRMDAT*, *IOPAR* and *PRTPAR*) can be stored with different contents in different directories. This allows you, for example, to save several sets of statistical data in different directories.

Reserved Variable	What It Contains	Used By
aENTER	Program run on ENTER.	Vectored ENTER
ALRMDAT	Alarm parameters.	TIME ALRM operations
βENTER	Program run after ENTER.	Vectored ENTER
CASDIR (directory)	Directory containing reserved CAS variables.	Computer Algebra System
CST	List defining the CST (custom) menu.	MENU, ← CUSTOM
"*der*"- *names*	User-defined derivative.	*∂*
EQ	Current equation.	ROOT, DRAW
EXITED	Program run after EDIT.	EDIT
EXPR	Current expression.	SYMBOLIC
IOPAR	I/O parameters.	I/O commands
MASD.INI	Valid source code.	Compiler
MHpar	Minehunt game status.	MINEHUNT
Mpar	Multiple-Equation Solver equations.	EQ LIB
n1, n2, …	Arbitrary integers.	ISOL, QUAD
Nmines	Minehunt game data.	MINEHUNT

Menus 0 through 117

These menus are mostly compatible with the menus in the HP 48G series.

```
  0 LAST MENU
  1 CUSTOM (no choose-box version available)
  2 VAR (no choose-box version available)
  3 MTH (or APPS 10: "MATH")
  4 MTH VECTR (MTH 1: "VECTOR")
  5 MTH MATRX (MTH 2: "MATRIX")
  6 MTH MATRX MAKE (MTH 2 1: "MATRIX MAKE")
  7 MTH MATRX NORM (MTH 2 2: "MATRIX NORM")
  8 MTH MATRX FACTR (MTH 2 3: "MATRIX FACTOR.")
  9 MTH MATRX COL (MTH 2 4: "CREATE COL")
 10 MTH MATRX ROW (MTH 2 5: "CREATE ROW")
 11 MTH LIST (MTH 3: "LIST")
 12 MTH HYP (MTH 4: "HYPERBOLIC")
 13 MTH NXT PROB (MTH 7: "PROBABILITY")
 14 MTH REAL (MTH 5: "REAL")
 15 [MTH] BASE (MTH 6 or BASE: "BASE")
 16 [MTH] BASE NXT LOGIC (MTH 6 7 or BASE 7: "LOGIC")
 17 [MTH] BASE NXT BIT (MTH 6 8 or BASE 8: "BIT")
 18 [MTH] BASE NXT BYTE (MTH 6 9 or BASE 9: "BYTE")
 19 MTH NXT FFT (MTH 8: "FFT")
 20 MTH NXT CMPLX (MTH 9: "COMPLEX")
 21 MTH NXT CONST (MTH 10: "CONSTANTS")
 22 PRG (PRG: "PROG")
 23 PRG BRCH (PRG 3: "BRANCH")
 24 PRG BRCH IF (PRG 3 1: "IF")
 25 PRG BRCH CASE (PRG 3 2: "CASE")
 26 PRG BRCH START (PRG 3 3: "START")
 27 PRG BRCH FOR (PRG 3 4: "FOR")
 28 TOOL EDIT (no choose-box version available)
 29 PRG BRCH DO (PRG 3 5: "DO")
* 30 old soft-menu solver (no choose-box version available)
 31 PRG BRCH WHILE (PRG 3 6: "WHILE")
 32 PRG TEST (PRG 4: "TEST")
 33 PRG TYPE (PRG 5: "TYPE")
 34 PRG LIST (PRG 6: "LIST")
 35 PRG LIST ELEM (PRG 6 1: "ELEMENT")
 36 PRG LIST PROC (PRG 6 2: "PROC")
 37 PRG NXT GROB (PRG 7: "GROB")
 38 PRG NXT PICT (PRG 8: "PICTURE")
 39 PRG NXT IN (PRG 11: "INPUT")
 40 PRG NXT OUT (PRG 12: "OUTPUT")
 41 PRG NXT NXT RUN (no choose-box version available)
 42 [CONVERT] UNITS (CONVERT 1 or UNITS: "UNITS")
 43 [CONVERT] UNITS LENG (CONVERT 1 2 or UNITS 2: "LENGTH")
 44 [CONVERT] UNITS AREA (CONVERT 1 3 or UNITS 3: "AREA")
 45 [CONVERT] UNITS VOL (CONVERT 1 4 or UNITS 4: "VOLUME")
 46 [CONVERT] UNITS TIME (CONVERT 1 5 or UNITS 5: "TIME")
 47 [CONVERT] UNITS SPEED (CONVERT 1 6 or UNITS 6: "SPEED")
 48 [CONVERT] UNITS NXT MASS (CONVERT 1 7 or UNITS 7: "MASS")
 49 [CONVERT] UNITS NXT FORCE (CONVERT 1 8 or UNITS 8: "FORCE")
 50 [CONVERT] UNITS NXT ENRG (CONVERT 1 9 or UNITS 9: "ENERGY")
```


Parameter (Command)	Description	Default Value
date (→DATE)	A real number specifying the date of the alarm: *MM.DDYYYY* (or *DD.MMYYYY* if flag –42 is set). If *YYYY* is not included, the current year is used.	Current date.
Time (→TIME)	A real number specifying the time of the alarm: *HH.MMSS*.	00.0000
action	A string or object: ■ a string creates an *appointment alarm*, which beeps and displays the string ■ any other object creates a *control alarm*, which executes the object	Empty string (appointment alarm).
Repeat	A real number specifying the interval between automatic recurrences of the alarm, given in ticks (a tick is $^{1}/_{8192}$ of a second).	0

Parameters without commands can be modified with a program by storing new values in the list contained in ALRMDAT (use the PUT command).

βENTER

This is the vectored ENTER post-processor. If flags –62 and –63 are set and ENTER is pressed, the command that triggered the command-line processing is put on the stack as a string and ßENTER is evaluated.

CST

CST contains a list (or a name specifying a list) of the objects that define the CST (*custom*) menu. Objects in the custom menu behave as do objects in built-in menus. For example:

- Names behave like the VAR menu keys. Thus, if *ABC* is a variable name, [ABC] evaluates ABC, [→] [ABC] recalls its contents, and [←] [ABC] stores new contents in *ABC*.
- The menu label for the name of a directory has a bar over the left side of the label; pressing the menu key switches to that directory.
- Unit objects act like unit catalog entries (and have left-shifted conversion capabilities, for example).
- String keys echo the string.
- You can include backup objects in the list defining a custom menu by tagging the name of the backup object with its port location.

In the first page of the PRG BRCH menu, pressing the shift keys before pressing any menu key provides a handy typing shortcut for programmers. In all these cases, the cursor is placed at the end of the first command. Thus these shifted keys can be thought of as "program structure delimiters". While entering a program,

◁ IF	types	IF THEN END
▷ IF	types	IF THEN ELSE END
◁ CASE	types	CASE THEN END END
▷ CASE	types	THEN END
◁ START	types	START NEXT
▷ START	types	START STEP
◁ FOR	types	FOR NEXT
▷ FOR	types	FOR STEP
◁ DO	types	DO UNTIL END
◁ WHILE	types	WHILE REPEAT END
◁ IFERR	types	IFERR THEN END
▷ IFERR	types	IFERR THEN ELSE END

In the EDIT menu, ▷ ⌫ and ▷ DEL perform skip-to-beginning and skip-to-end of line, respectively, and ▷ ⌫ and ▷ DEL perform delete-to-beginning and delete-to-end of line, respectively. ▷ GOTO is a shortcut for GOTO LABEL.

In the units menus, pressing a unit menu key multiplies by that unit, whereas ▷ UNIT divides by that unit. ◁ UNIT converts to that unit, if possible.

In program mode, ▷ UNIT returns `'1_unit' /`, while ◁ UNIT returns `'1_unit' CONVERT`, where "unit" is the unit corresponding to the menu key.

Parameter (Command)	Description	Default Value
baud (BAUD)	The baud rate: 2400, 4800, 9600, 14400, 19200, 38400 or 115200.	115200
parity (PARITY)	The parity used: 0=none, 1=odd, 2=even, 3=mark, 4=space. The value can be positive or negative: a positive parity is used upon both transmit and receive; a negative parity is used only upon transmit.	0
receive pacing	Controls whether receive pacing is used: a nonzero real value enables pacing, while zero disables it. Receive pacing sends an XOFF signal when the receive buffer is almost full, and sends an XON signal when it can take more data again. Pacing is not used for Kermit I/O, but is used for other serial I/O transfers.	0 (no pacing)
transmit pacing	Controls whether transmit pacing is used: a nonzero real value enables pacing, while zero disables it. Transmit pacing stops transmission upon receipt of XOFF, and resumes transmission upon receipt of XON. Pacing is not used for Kermit I/O, but is used for other serial I/O transfers.	0 (no pacing)
checksum (CKSM)	Error-detection scheme requested when initiating SEND: ■ 1=1-digit arithmetic checksum ■ 2=2-digit arithmetic checksum ■ 3=3-digit cyclic redundancy check.	3
translation code (TRANSIO)	Controls which characters are translated: ■ 0=none ■ 1=translate character 10 (line feed only) to / from characters 10 and 13 (line feed and carriage return) ■ 2=translate characters with numbers 128 through 159 (80-9F hex) ■ 3=translate characters with numbers 128 through 255.	1

Parameters without commands can be modified with a program by storing new values in the list contained in *IOPAR* (use the PUT command), or by editing *IOPAR* directly.

Other keyboard shortcuts

Keycode	Keystroke	Definition
11.21	↰ & F1 through F6	Accesses the graphing input forms
22.21	↰ & CUSTOM (↰ & MODE)	MODES menu (menu 63)
23.21	↰ & i (↰ & TOOL)	Toggles real/complex mode (flag –103)
25.1	▲	If no command line, interactive stack (same as HIST key)
31.21	↰ & UPDIR (↰ & VAR)	HOME
33.21	↰ & PREV (↰ & NXT)	Last Menu
34.1	◀	PICTURE
35.1	▼	EDITB
35.2	↰ ▼	VISIT for variables, EDIT for everything else.
35.21	↰ & ▼	VISITB for variables, EDITB for everything else.
35.3	↱ ▼	If the current menu has a special info-screen, this displays it; otherwise, the menu itself is displayed full-screen, with the VAR menu also showing the beginning of each object.
36.1	▶	If no command line, performs SWAP in RPN mode
36.3	↱ ▶	If no command line, XSERV
36.31	↱ & ▶	If no command line, SERVER
42.31	↱ & CHARS (↱ & EVAL)	CHARS menu (menu 62)
43.31	↱ & EQW (↱ & ')	` ` (places back-tics on the command line)
43.61	ALPHA ↱ & '	Ω (omega)
45.1	⌫	If no command line, performs DROP in RPN mode
45.2	↰ DEL (↰ ⌫)	If no command line, clears the stack
45.3	↱ CLEAR (↱ ⌫)	If no command line, clears the stack
62.1	+/–	Toggles through available values when cursor is on a choose field. For example, MODE +/– will change operating modes from RPN to algebraic, or algebraic to RPN.
72.5 through 74.5	ALPHA ↰ 7, 8, or 9	Modifies the most recent letter on the command line to add a diacritical mark to the character.
72.31	↱ & NUM.SLV (↱ & 7)	Solver menu (menu 74)
74.31	↱ & TIME (↱ & 9)	TIME menu (menu 167)
84.61	ALPHA ↱ & 6	° (degree symbol)

Parameter (Command)	Description	Default Value
(x_{min}, y_{min}) (XRNG, YRNG)	A complex number specifying the lower left corner of *PICT* (the lower left corner of the display range).	(–6.5, –3.1) 48gII (–6.5, –3.9) 49g+ (–6.5, –3.9) 50g
(x_{max}, y_{max}) (XRNG, YRNG)	A complex number specifying the upper right corner of *PICT* (the upper right corner of the display range).	(6.5, 3.2) 48gII (6.5, 4.0) 49g+ (6.5, 4.0) 50g
indep (INDEP)	A name specifying the independent variable, or a list containing that name and two numbers that specify the minimum and maximum values for the independent variable (the plotting range).	*X*
res (RES)	Resolution. A real number specifying the interval between values of the independent variable. For plots of equations, this determines the plotting interval along the *x*-axis. A binary number specifies the *pixel* resolution (how many columns of pixels between points). An integer specifies the resolution in *user* units (how many user units between points). Resolution for statistical plots is different (see below).	0
axes (AXES)	A complex number specifying the user-unit coordinates of the plot origin, or a list containing the following: ■ the complex number specifying the origin ■ a real number, binary integer, or list containing two real numbers or binary integers specifying the tick-mark annotation (see ATICK) ■ two strings specifying labels for the horizontal and vertical axes	(0, 0)
ptype (BAR, etc.)	A command name specifying the plot type (BAR, CONIC, DIFFEQ, FAST3D, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, or YSLICE).	FUNCTION
depend (DEPND)	A name specifying the dependent variable, or a list containing the name and two numbers that specify vertical plotting range. For DIFFEQ, the second element of the list may also be a real vector that represents the initial value.	*Y*

Parameters without commands can be modified with a program by storing new values in the list contained in *PPAR* (use the PUT command).

■ **_ (Attach Unit).** This command will create unit objects in parallel only if level 1 contains a list. Thus `1 { ft in m } _` produces `{ 1_ft 1_in 1_m }` while `{ 1 2 3 } 'm' _` produces an error.

■ **STO+.** STO+ performs parallel list addition only if both arguments are lists. If one argument is a list and the other is not, STO+ appends the non-list argument to each element in the list.

■ **STO-, STO*, STO/.** These commands perform parallel processing if both arguments are lists, but fail otherwise.

Using DOLIST for Parallel Processing

Almost any command or user program can be made to work in parallel over a list or lists of data y using the DOLIST command. Use DOLIST as follows.

■ Level 1 must contain a command, a program object, or the name of a variable that contains a command or program object.

■ Level 2 must contain an argument count unless the level 1 object is a command that accepts parallel processing, or a user-defined function. In these special cases, Level 2 contains the first of the list arguments.

■ If level 2 was the argument count, then level 3 is the first of the argument lists. Otherwise, levels 2 through are the argument lists.

As an example, the following program takes three objects from the stack, tags them with the names `a`, `b`, and `c`, and displays them one after the other in line 1 of the display.

```
« → a b c
 « { a b c } DUP « EVAL » DOLIST
  SWAP « →TAG » DOLIST
  CLLCD 1 « 1 DISP 1 WAIT » DOLIST
 »
»
```


STARTEQW

If it exists, the STARTEQW variable is evaluated whenver an element in the Equation Writer is selected.

STARTERR

If it exists, the STARTERR variable is evaluated whenever an error message is displayed.

STARTOFF

If it exists, the STARTOFF variable is evaluated when the calculator turnsoff automatically.

STARTRECV

If it exists, the STARTRECV variable is evaluated when the user presses RECV from inside the Filer. It takes as an argument, the selected argument's name.

STARTSEND

If it exists, the STARTSEND variable is evaluated when the user presses SEND from inside the Filer. It takes as arguments the selected object and its name.

STARTUP

If it exists, the STARTUP variable is evaluated when the calculator warm starts. Objects placed on the stack in the STARTUP routine cannot be expected to remain on the stack after the calculator starts.

s1, s2, …

The ISOL and QUAD commands return *general* solutions (as opposed to *principal* solutions) for operations. A general solution contains variables for arbitrary integers or arbitrary signs or both.

The variable *s1* represents an arbitrary + or – sign. Additional arbitrary signs are represented by *s2*, *s3*, etc.

If flag –1 is set, the ISOL and QUAD return principal solutions, in which case the arbitrary sign is always +1.

TOFF

TOFF contains a binary integer #xxxxh that defines the number of clock ticks until the next calculator auto shutoff (with a default of 5 minutes or 6*60*8192 ticks)

integer matrix having 3 rows and 4 columns. Since these commands can normally use lists as arguments, they cannot perform parallel processing, except by using DOLIST.

■ **Program control commands.** Program control structures and commands do no perform parallel processing and cannot be forced to do so. However, programs containing these structures can be made to parallel process by using DOLIST. For example, `{ 1 2 3 4 5 6 } 4 « IF DUP 3 ≥ THEN DROP END » DOLIST` returns `{ 3 4 5 6 }`.

Group 3: commands that sometimes work with parallel processing

■ Graphics commands that can take pixel coordinates as arguments expect those coordinates to be presented as two-element lists of binary integers. Since these commands can normally use lists as arguments, they cannot parallel process, except by using DOLIST.

■ For the two-argument graphics commands (BOX, LINE, TLINE), if either argument is not a list (a complex number, for example), then the commands will parallel process, taking the list argument to be multiple complex number coordinates. For example, `(0,0) {(1,1) (3,2)} LINE` will draw two lines — between (0,0) and (1,1) and between (0,0) and (3,2).

Group 4: ADD and +

■ On HP 48S and HP 48SX calculators, the + command has been used to append lists or to append elements to lists. Thus `{ 1 2 3 } 4 +` returns `{ 1 2 3 4 }`. With the advent of parallel processing in the HP 48G series, the ADD command was created to perform parallel addition instead of +.

This has several ramifications:

■ To add two lists in parallel, you must do one of the following:

- ▢ Use ADD from the ◀ MTH ▶ menu.
- ▢ Create a custom menu containing the ADD command.
- ▢ Assign the ADD command to a user-defined key.

■ User programs must be written using ADD instead of + if the program is to be able to perform direct parallel processing, or written with + and applied to their arguments by using DOLIST. For example, programs such as `« →x 'x+2' »` will produce list concatenation when x is a list rather than parallel addition, unless rewritten as `« → x 'x ADD 2' »`

■ Algebraic expressions capable of calculating with variables containing lists (including those intended to become user-defined functions) cannot be created in RPN syntax since using ADD to add two symbolic arguments concatenates the arguments with + rather than with ADD. For example, `'X' DUP 2 ^ SWAP 4 * ADD 'F(X)' SWAP =` produces `'F(X)=X^2+4*X'` rather than `'F(X)=X^2 ADD 4*X'`.

Group 5: Commands that set modes / states

Commands that store values in system-specific locations so as to control certain modes and machine states can generally be used to parallel process data. The problem is that each successive parameter in the list cancels the setting established by the previous parameter. For example, `{ 1 2 3 4 5 } FIX` is effectively the same as `5 FIX`.

Group 6: One-argument, one-result commands

These commands are the easiest to use with parallel processing. Simply provide the command with a list of arguments instead of the expected single argument. Some examples:

`{ 1 -2 3 -4 }` returns `{ 1 2 3 4 }`

`DEG { 0. 30. 60. 90. } SIN` returns `{ 0. .5 .866025403784 1. }`

`{ 1 A 'SIN(Z)' } INV` returns `{ 1 'INV(A)' 'INV(SIN(Z))' }`

ZPAR

ZPAR is a variable in the current directory. It contains a list of zooming parameters used by the DRAW command for all 2-D mathematical and statistical plots.

Parameter (Command)	Description	Default Value
h-factor	Real number that specifies the horizontal zoom factor.	4
v-factor	Real number that specifies the vertical zoom factor.	4
recenter flag	0 or 1 depending on whether the recenter at crosshairs option was selected in the set zoom factors input form.	0
{ *list* }	An empty list, or a copy of the last *PPAR*.	

Use the set zoom factors input form (▶) to modify *ZPAR*.

ΣDAT

ΣDAT is a variable in the current directory that contains either the current statistical matrix or the name of the variable containing this matrix. This matrix contains the data used by the Statistics applications.

var_1	var_2	…	var_m
x_{11}	x_{21}	…	x_{m1}
x_{12}	x_{22}	…	x_{m2}
⋮	⋮	⋮	⋮
x_{1n}	x_{2n}	…	x_{mn}

Statistical Matrix for Variables 1 to m

You can designate a new current statistical matrix by entering new data, editing the current data, or selecting another matrix.

The command CLΣ clears the current statistical matrix.

Source References

The following references were used as sources for many of the constants and equations used in the calculator. (See "References" in chapter 5, "Equation Reference," for the references used as sources for the Equation Library.)

1. E.A. Mechtly. *The International System of Units, Physical Constants and Conversion Factors*, Second Revision. National Aeronautics and Space Administration, Washington DC, 1973.
2. *The American Heritage Dictionary.* Houghton Mifflin Company, Boston, MA, 1979.
3. *American National Standard Metric Practice ANSI/IEEE Std 268-1982.* The Institute of Electrical and Electronics Engineers, Inc., New York, 1982.
4. *ASTM Standard Practice for Use of the International System of Units (SI) E380-89a.* American Society for Testing and Materials, Philadelphia, 1989.
5. *Handbook of Chemistry and Physics*, 64th Edition, 1983-1984. CRC Press, Inc, Boca Raton, FL, 1983.
6. *International Standard publication No. ISO 31/1-1978 (E).*
7. *The International System of Units (SI)*, Fourth Edition. The National Bureau of Standards Special Publication 330, Washington D.C., 1981.
8. *National Aerospace Standard.* Aerospace Industries Association of America, Inc., Washington D.C., 1977.
9. *Physics Letters B,* vol 204, 14 April 1988 (ISSN 0370-2693).

CASDIR Reserved Variables

The Computer Algebra System stored its reserved variables in a special directory called CASDIR.

Reserved Variable	What It Contains	Used By
CASINFO	Graphic display temporary storage.	Step-by-step operations
ENVSTACK	Flags and current path.	PUSH, POP
EPS	Maximum coefficient to round to zero.	EPSX0
IERR	Error of integration	Numerical integration operations
MODULO	The current modulus.	MOD functions
PERIOD	Period for periodic operations.	Various CAS operations
PRIMIT	Anti-derivative temporary storage.	Various CAS operations
REALASSUME	List of variables to treat as real numbers in complex mode.	Various CAS operations
VX	Default variable for symbolic operations.	Various CAS operations

Contents of the CASDIR Reserved Variables

CASINFO

CASINFO provides temporary storage for the graphic display during step-by-step operations.

ENVSTACK

ENVSTACK is a variable stored in the CAS directory. It is used by PUSH and POP to save the status of flags and the current directory. (PUSH saves the data in *ENVSTACK*; POP restores it.)

EPS

EPS contains a real number specifying that coefficients in a polynomial smaller than this value are replaced with 0. It is used by the EPSX0 command. The default value is 1E-10.

IERR

IERR contains the error tolerance of integration during numeric integration.

Trigonometric Expansions

The following tables list expansions for trigonometric functions in Radians mode when using the →DEF, TRG*, and →TRG operations. These operations appear in the Equation Writer RULES menu.

→DEF Expansions

Function	Expansion
SIN (*x*)	$\frac{EXP(x \times i) - EXP(-(x \times i))}{2 \times i}$
COS (*x*)	$\frac{EXP(x \times i) + EXP(-(x \times i))}{2 \times i}$
TAN (*x*)	$\frac{EXP(x \times i \times 2) - 1}{(EXP(x \times i \times 2) + 1) \times i}$
SINH(*x*)	$-(SIN(x \times i) \times i)$
COSH(*x*)	$COS(x \times i)$
TANH(*x*)	$TAN(x \times i) \times -i$
ASIN(*x*)	$-i \times LN(\sqrt{1 - x^2} + i \times x)$
ACOS(*x*)	$\frac{\pi}{2} + i \times LN(\sqrt{1 - x^2} + i \times x)$
ATAN(*x*)	$-i \times LN\left(\frac{(1 + i \times x)}{\sqrt{1 + x^2}}\right)$
ASINH(*x*)	$-LN(\sqrt{1 + x^2} - x)$
ACOSH(*x*)	$\sqrt{-\left(\frac{\pi}{2} + i \times LN(\sqrt{1 - x^2} + i \times x)\right)^2}$
ATANH(*x*)	$-LN\left(\frac{(1 - x)}{\sqrt{1 - x^2}}\right)$

TRG* Expansions

Function	Expansion
SIN (*x* + *y*)	$SIN(x) \times COS(y) + COS(x) \times SIN(x)$
COS (*x* + *y*)	$COS(x) \times COS(y) + SIN(x) \times SIN(y)$
TAN (*x* + *y*)	$\frac{TAN(x) + TAN(y)}{1 - TAN(x) \times TAN(y)}$
SINH(*x* + *y*)	$SINH(x) \times COSH(y) + COSH(x) \times SINH(x)$
COSH(*x* + *y*)	$COSH(x) \times COSH(y) + SINH(x) \times SINH(y)$
TANH(*x* + *y*)	$\frac{TANH(x) + TANH(y)}{1 + TANH(x) \times TANH(y)}$

→TRG Expansions

Function	Expansion
EXP (*x*)	$COS\left(\frac{x}{i}\right) + SIN\left(\frac{x}{i}\right) \times i$

Technical Reference

This appendix contains the following information:

- Object sizes.
- Symbolic integration patterns used by the calculator.
- Trigonometric expansions used by the calculator.
- Precedence of operations.
- References and as sources for constants and equations in the calculator (other than those in the Equation Library).

Object Sizes

The following table lists object types and their size in bytes. (Note that characters in names, strings, and tags use 1 byte each.)

Object Size

Object Type	Size (bytes)
Algebraic	5 + size of included objects
Backup Object	12 + number of name characters + size of included object
Binary Integer	13
Command	2.5
Complex matrix	15 + 16 × number of elements
Complex number	18.5
Complex vector	12.5 + 16 × number of elements
Directory	6.5 + size of included variables + size of variable names + 2.5 bytes for header
Graphics Object	10 + number of rows × CEIL (columns/8)
Integer	2.5 or (5 + 0.5 × number of digits)
List	5 + size of included objects
Matrix	15 + 8 × number of elements
Program	10 + size of included objects
Quoted global or local name	8.5 + number of characters
Real number	2.5 or 10.5
String	5 + number of characters
Tagged Object	3.5 + number of tag characters + size of untagged object

Command	Type	Library	Size	Keys	Menu	?	First
YSLICE	C	171-11	5.5	85 MENU	85 2219.02		48GX
YVOL	C	171-1	5.5	86 MENU	86 2219		48GX
YYRNG	C	171-4	5.5	86 MENU	86 2219		48GX
ZEROS	C	788-64	5.5	SYMB SOLVE S.SLV NXT «SOLVER»	120.02 142.03 154	H	1.05
ZFACTOR	F	171-95	5.5	APPS 12 UTILS 117 MENU	117 2219.16		48GX
ZVOL	C	171-2	5.5	86 MENU	86 2219		48GX
^	A	2-73	2.5	key 51.1: y^x			28C
_	F	2-267	2.5	key 85.3			28C
dB	F	171-105	5.5	APPS 12 UTILS NXT 117.02 MENU	117.02 2219.18		48GX
e	F	2-66	2.5	ALPHA-LS-E MTH NXT CONST «CONSTANTS»	21		28C
gmol	F	171-102	5.5	APPS 12 UTILS NXT 117.02 MENU	117.02 2219.18		48GX
i	F	2-67	2.5	key 23.2 CMPLX MTH NXT CONST «CMPLX» «CONSTANTS»	21 130		28C
lbmol	F	171-103	5.5	APPS 12 UTILS NXT 117.02 MENU	117.02 2219.18		48GX
lim	F	788-5	5.5	SYMB CALC CALC LIMIT «DIFF» NXT	136.03 152 163	H	1.20
qr	C	222-8	5.5	MATRICES FACT	148	H	1.20
rpm	F	171-104	5.5	APPS 12 UTILS NXT 117.02 MENU	117.02 2219.18		48GX
rref	C	788-71	5.5	SYMB SOLVE MATRICES LIN-S «MATR»	154 158	H	1.05
|	F	2-255 2-256	2.5	key 23.3 «ALGB» NXT 93.02 MENU	93.02		28C
√	A	2-79	2.5	key 52.1			28C
∫	F	2-252 2-253	2.5	key 55.3			28C
Σ	F	2-254	2.5	key 53.3			28C
Σ+	C	2-286	2.5	91/97 MENU	91 97		28C
Σ-	C	2-287	2.5	91/97 MENU	91 97		28C


```
154 SYMB SOLVE (SYMB 5: "SYMBOLIC SOLVER")
155 SYMB NXT EXPLN (SYMB 7: "SYMBOLIC EXP & LN")
156 MATRICES OPER (MATRICES 2: "MATRIX OPERATIONS")
157 MATRICES QUADF (MATRICES 4: "MATRIX QUAD. FORM")
158 MATRICES LIN-S (MATRICES 5: "MATRIX LINEAR SYS.")
159 MATRICES NXT EIGEN (MATRICES 7: "MATRIX EIGENVECT.")
160 MATRICES NXT VECT (MATRICES 8: "MATRIX VECTOR")
161 TRIG HYP (TRIG 1: "TRIG HYPERBOLIC")
162 CALC DERIV (CALC 1: "DERIV. & INTEG.")
163 CALC LIMIT (CALC 2: "LIMITS & SERIES")
164 CALC DIFF (CALC 3: "DIFFERENTIAL EQNS")
165 MATRICES CREAT COL (MATRICES 1 1: "CREATE COL")
166 MATRICES CREAT ROW (MATRICES 1 2: "CREATE ROW")
167 APPS TIME TOOLS (APPS 5 4: "TIME")
168 CONVERT BASE (CONVERT 2: "BASE")
169 CONVERT BASE NXT LOGIC (CONVERT 2 3: "LOGIC")
170 CONVERT BASE NXT BIT (CONVERT 2 4: "BIT")
171 CONVERT BASE NXT BYTE (CONVERT 2 5: "BYTE")
172 CONVERT REWRITE (CONVERT 4: "REWRITE")
173 CONVERT MATRIX (CONVERT 5: "MATRIX CONVERT")
174 ARITH PERM (ARITH 4: "PERMUTATION")
175 MATRICES LINAP (MATRICES 6: "LINEAR APPL")
176 MTH SPECIAL (MTH 11: "SPECIAL FUNCTIONS")
177 CALC GRAPH (CALC 4: "SYMBOLIC GRAPH")
```

Menus 178 through 255

These menus do not exist and are available for future use by the operating system.

Built-In Library Menus

The menus for built-in libraries with library numbers of 256 and higher can be accessed directly with their library number. Libraries with library numbers under 256 can be accessed by adding 2048 to the library number.

```
 256 Library 256: Development Library (not attached by default)
     Note: Warmstart with B C and D held down forces "smart" mode, in which library 256
     is attached and RPN mode is the default.  When library 256 is attached, it is added
     to the bottom of the APPS menu.
 257 Library 257: MASD V5.2 Assembler (used internally by library 256; do not use)
 258 Library 256: extable (not installed by default)
     Note: Stores mnemonics for development with library 256.  Must be downloaded
     and installed separately.
 788 Library 788: Computer Algebra System, an improved version of the Erable library
1792 Library 1792: Program structure commands (same as in 48S/G)
2050 Library 2: 48S command set + new ones
2057 Library 9: Statistical test functions (do not use)
2219 Library 171: 48G command set + new ones
2269 Library 221: Meta Kernel commands (not in any menus)
2270 Library 222: New commands & CAS messages
2275 Library 227: Equation Library + MINEHUNT (new in Version 2.00)
2277 Library 229: Periodic Table Library (new in Version 2.15)
2289 Library 241: Statistics commands
```

Command	Type	Library	Size	Keys	Menu	?	First
TRIGSIN	C	788-29	5.5	TRIG NXT NXT CONVERT TRIG NXT «TRIGO» NXT NXT	122.03 140.02	H	1.05
TRIGTAN	C	788-30	5.5	TRIG NXT NXT «TRIGO» NXT NXT	122.03	H	1.05
TRN	C	2-175	2.5	MTH MATRX MAKE	6		28C
TRNC	F	2-109	2.5	MTH REAL NXT NXT	14.03		48SX
TRUNC	F	788-99	5.5	«DIFF» NXT NXT 99 DUP MENUXY		H	1.05
TRUTH	C	2-224	2.5	82 MENU	82		48SX
TSIMP	C	788-21	5.5	EXP&LN NXT TRIG NXT NXT CONVERT TRIG NXT NXT «EXP&LN»	121.02 122.03 140.03	H	1.05
TSTR	C	2-29	2.5	RS&TIME NXT NXT PRG NXT NXT TIME NXT NXT APPS 5 4 NXT NXT	94.03 167.03		48SX
TVARS	C	2-37	2.5	PRG MEM DIR NXT	71.02		48SX
TVM	C	171-71	5.5	RS&NUM.SLV TVM SOLVR	80 2219.12		48GX
TVMBEG	C	171-72	5.5	CAT RS&NUM.SLV TVM LS-BEG	2219.13		48GX
TVMEND	C	171-73	5.5	CAT RS&NUM.SLV TVM RS-BEG	2219.13		48GX
TVMROOT	C	171-74	5.5	RS&NUM.SLV TVM	79 2219.13		48GX
TYPE	C	2-166	2.5	PRG TEST NXT PRG TYPE NXT NXT	32.02 33.03		28C
UBASE	F	2-14	2.5	[CONVERT] UNITS TOOLS	59 118		48SX
UFACT	C	2-15	2.5	[CONVERT] UNITS TOOLS	59 118		48SX
UFL1→MINIF	C	221-20	5.5	CAT	2269.04		1.05
UNASSIGN	F	222-49	5.5	«ALGB» NXT		H	1.20
UNASSUME	F	222-39	5.5	«TESTS»		H	1.20
UNBIND	C	222-61	5.5	CAT		H	1.20
UNPICK	C	2-395	2.5	PRG/TOOL STACK NXT	73.02		1.05
UNROT	C	2-394	2.5	PRG/TOOL STACK	73		1.05
UNTIL	C	1792-8	2.5	PRG BRCH DO	29 145.02		28C
UPDIR	C	2-35	2.5	key 31.2			48SX
UTPC	C	2-308	2.5	MTH NXT PROB NXT			28C
UTPF	C	2-310	2.5	MTH NXT PROB NXT			28C
UTPN	C	2-309	2.5	MTH NXT PROB NXT			28C
UTPT	C	2-311	2.5	MTH NXT PROB NXT			28C
UVAL	F	2-12	2.5	[CONVERT] UNITS TOOLS	59 118		48SX

The Command Menu-Path Table

This lists the calculator's programmable commands in CAT order. Of the 818 commands, 808 are shown by CAT. This list assumes that library 256 is attached, flag –95 is off, and flag –117 is set.

"Keys" column:

Each command identifies its menu path or key sequence (if any) with alternatives on additional lines. "/" means "either". The most efficient key sequence is shown first if several exist, assuming that NXT NXT is better than PREV, etc. "[]" = optional.

! **key ALPHA-RS-2; MTH NXT PROB**

This means that you can either press [ALPHA] [RIGHT-SHIFT] [2] or press [MTH] [NXT] [PROB] [!] to get the ! function.

+ **key 95.1**

This means that [+] is on the keyboard in row 9, column 5. The
.1 means unshifted;
.2 means left-shifted;
.3 means right-shifted;
.4 means alpha-shifted;
.5 means alpha-left-shifted; and
.6 means alpha-right-shifted.
.01 added means hold down the shift key while pressing the key, e.g. key 22.21 is MODE with the left-shift key held down, also called LS&MODE.

AMORT **79 MENU**

This means that AMORT is not in any keyboard menu, but you can find it in numbered menu 79 (type 79 MENU to go to that menu).

QUOT **«POLYNOMIAL» NXT; ARITH POLY PREV**

This means that QUOT is in the POLYNOMIAL menu, a special menu which is neither a numbered menu, nor on the keyboard, but is seen by executing the «programmable» command POLYNOMIAL. QUOT can also be found in the ARITH POLY PREV menu.

Other columns:

Type: "F" is function, "C" is command, and "A" is analytic function.

Library: The internal library number and command number identifying the command. If there are multiple, they are listed one per line.

Size: Amount of memory taken in a program by this command, in bytes

Menu: The menu ID containing this command, accessible with the MENU command. If there are multiple, they are listed one per line.

?: An "H" means this command has a HELP screen. An "-" means "not shown in the CATalog".

First: For identifying backwards compatibility, this is the first calculator model (28C, 28S, 48SX, 48GX) or ROM version (49G through the 50g) with the command.

Command	Type	Library	Size	Keys	Menu	?	First
SUBTMOD	F	788-111	5.5	ARITH MODUL NXT «MODULAR» NXT	128.02	H	1.05
SVD	C	171-46	5.5	MATRICES FACT MTH MATRX FACTR	8 148 2219.08		48GX
SVL	C	171-47	5.5	MATRICES FACT NXT MTH MATRX FACTR NXT	8.02 148.02 2219.08		48GX
SWAP	C	2-271	2.5	PRG/TOOL STACK	73		28C
SYLVESTER	C	788-78	5.5	MATRICES QUADF «MATR» NXT	157	H	1.05
SYSEVAL	C	2-49	2.5	CAT			28C
SYST2MAT	C	222-10	5.5	CONVERT MATRX MATRICES LIN-S	158 173	H	1.20
S~N	C	256-22	5.5	256.04 MENU	256.04		1.05
S→H	C	256-8	5.5	256.02 MENU	256.02		1.05
TABVAL	C	788-97	5.5	SYMB GRAPH NXT CALC GRAPH NXT	142.02 153.02 177.02	H	1.05
TABVAR	C	788-96	5.5	SYMB GRAPH NXT CALC GRAPH NXT «DIFF» NXT	142.02 153.02 177.02	H	1.05
TAIL	C	171-82	5.5	RS&CHARS NXT PRG LIST ELEM NXT PRG NXT CHARS NXT	35.02 62.02 2219.14		48GX
TAN	A	2-83	2.5	key 55.1			28C
TAN2CS2	C	222-28	5.5	«TRIGO» NXT		H	1.20
TAN2SC	C	788-31	5.5	SYMB TRIG TRIG NXT CONVERT TRIG NXT «TRIGO» NXT	122.02 139 140.02	H	1.05
TAN2SC2	C	788-33	5.5	SYMB TRIG TRIG NXT CONVERT TRIG NXT «TRIGO» NXT	122.02 139 140.02	H	1.05
TANH	A	2-86	2.5	MTH HYP TRIG HYP «HYPERBOLIC»	12 161		28C
TAYLOR0	F	788-6	5.5	CALC LIMIT SYMB CALC NXT «DIFF» NXT	136.04 152.02 163	H	1.05
TAYLR	C	2-339	2.5	CALC LIMIT	93 163		28C
TCHEBYCHEFF	F	788-91	5.5	«POLYNOMIAL» NXT 91 DUP MENUXY		H	1.05
TCOLLECT	C	788-26	5.5	TRIG NXT «TRIGO» NXT	122.02	H	1.05
TDELTA	F	171-100	5.5	APPS 12 UTILS NXT 117.02 MENU	117 2219.17		48GX

Command	Type	Library	Size	Keys	Menu	?	First
ADD	C	171-92	5.5	MTH LIST	11 2219.16		48GX
ADDTMOD	F	788-110	5.5	ARITH MODUL «MODULAR»	128	H	1.05
ADDTOREAL	C	222-0	5.5	CAT		H	1.05
ALGB	C	788-128	5.5	«MAIN»		H	1.20
ALOG	A	2-96	2.5	key 61.2: 10^x			28C
AMORT	C	171-75	5.5	RS&NUM.SLV TVM	79 80 2219.13		48GX
AND	F	2-229	2.5	PRG TEST NXT [MTH/CONVERT] BASE NXT LOGIC «TESTS» NXT	16 32.02 169		28C
ANIMATE	C	171-20	5.5	PRG NXT GROB NXT	37.02 2219.04		48GX
ANS	C	2-387	2.5	key 105.2 in ALG mode CAT in RPL mode			1.05
APEEK	C	256-18	5.5	256.04 MENU	256.04		1.05
APPLY	F	2-258 2-259	2.5	93.02 MENU	93.02		48SX
ARC	C	2-216	2.5	PRG NXT PICT	38		48SX
ARCHIVE	C	2-353	2.5	PRG MEM NXT	70.02		48SX
ARG	F	2-77	2.5	key 65.3 CMPLX MTH NXT CMPLX «CMPLX»	20 130		48SX
ARIT	C	788-133	5.5	«MAIN» NXT		H	1.05
ARM→	C	256-34	5.5	256.06 MENU	256.06		2.00
ARRY→	C	2-171	2.5	CAT			28C
ASIN	A	2-87	2.5	key 53.2			28C
ASIN2C	C	788-36	5.5	TRIG CONVERT TRIG «TRIGO»	122 140	H	1.05
ASIN2T	C	788-35	5.5	TRIG CONVERT TRIG «TRIGO»	122 140	H	1.05
ASINH	A	2-90	2.5	MTH HYP TRIG HYP «HYPERBOLIC»	12 161		28C
ASM	C	256-30	5.5	256.06 MENU	256.06		1.16
ASM→	C	256-24	5.5	256.05 MENU	256.05		1.05
ASN	C	2-380	2.5	LS&MODE KEYS PRG NXT MODES KEYS	67		48SX
ASR	C	2-0	2.5	[MTH/CONVERT] BASE NXT BIT	17 170		28C
ASSUME	F	222-38	5.5	«TESTS»		H	1.20
ATAN	A	2-89	2.5	key 55.2			28C

Command	Type	Library	Size	Keys	Menu	?	First
SIN	A	2-81	2.5	key 53			28C
SINCOS	C	788-24	5.5	TRIG NXT	122.02	H	1.05
				CONVERT TRIG	140		
				SYMB NXT EXPLN	155		
				«TRIGO»			
				«REWRITE»			
SINH	A	2-84	2.5	MTH HYP	12		28C
				TRIG HYP	161		
				«HYPERBOLIC»			
SINV	C	2-324	2.5	PRG MEM ARITH NXT	72.02		28C
SIZE	C	2-160	2.5	RS&CHARS	6		28C
				MTH MATRX MAKE	35		
				PRG LIST ELEM	37.02		
				PRG NXT CHARS	62		
				PRG NXT GROB NXT	156.03		
				MATRICES OPER NXT NXT			
SL	C	2-5	2.5	[MTH/CONVERT] BASE NXT BIT	17		28C
					170		
SLB	C	2-6	2.5	[MTH/CONVERT] BASE NXT BYTE	18		28C
					171		
SLOPEFIELD	C	171-12	5.5	85 MENU	85		48GX
					2219.03		
SNEG	C	2-325	2.5	PRG MEM ARITH NXT	72.02		28C
SNRM	C	171-41	5.5	MTH MATRX NORM	7		48GX
				MATRICES OPER NXT NXT	147.02		
					156.03		
					2219.07		
SOLVE	C	788-63	5.5	S.SLV	120	H	1.05
				SYMB SOLVE	124.02		
				ALG NXT	142.02		
				«SOLVER»	154		
SOLVEQN	C	227-1	5.5	APPS 12 EQLIB	2275.01		48GX
				114 MENU			
SOLVER	C	788-134	5.5	«MAIN»		H	1.05
SOLVEVX	C	788-8	5.5	S.SLV	120	H	1.05
				SYMB SOLVE	142.02		
				«SOLVER»	154		
SORT	C	171-94	5.5	MTH LIST	11		48GX
				PRG LIST PROC NXT	36.02		
					2219.16		
SPHERE	C	171-19	5.5	LS&MODE ANGLE	4.02		48GX
				MTH VECTR NXT	65		
				PRG NXT MODES ANGLE	2219.04		
SQ	A	2-80	2.5	key 52.2: x^2			28C
SR	C	2-7	2.5	[MTH/CONVERT] BASE NXT BIT	17		28C
					170		
SRAD	C	171-40	5.5	MTH MATRX NORM	7		48GX
				MATRICES OPER NXT NXT	147.02		
					156.03		
					2219.07		

Command	Type	Library	Size	Keys	Menu	?	First
CF	C	2-132	2.5	LS&MODE FLAG	32.03		28C
				PRG TEST NXT NXT	66		
				PRG NXT MODES FLAG			
CHINREM	C	788-58	5.5	ARITH POLY	126	H	1.05
CHOLESKY	C	222-11	5.5	MATRICES QUADF	157	H	1.20
CHOOSE	C	171-77	5.5	PRG NXT IN	39		48GX
					2219.13		
CHR	C	2-165	2.5	RS&CHARS	33.02		28C
				PRG TYPE NXT	62		
				PRG NXT CHARS			
CIRC	C	222-29	5.5	ARITH PERM	174	H	1.20
CKSM	C	2-370	2.5	106 MENU	106		48SX
CLEAR	C	2-282	2.5	key 45.3			28C
CLKADJ	C	2-24	2.5	RS&TIME NXT NXT	94.03		48SX
				PRG NXT NXT TIME NXT NXT	167.03		
				APPS 5 4 NXT NXT			
CLLCD	C	2-56	2.5	PRG NXT OUT	40		28C
CLOSEIO	C	2-363	2.5	104.02 MENU	104.02		48SX
CLUSR	C	2-347	2.5	alias for CLVAR		-	28C
CLVAR	C	2-347	2.5	CAT			48SX
CLΣ	C	2-284	2.5	91/97 MENU	91		28C
					97		
CMPLX	C	788-129	5.5	«MATHS»		H	1.05
				«MAIN» NXT			
CNRM	C	2-119	2.5	MATRICES OPER	7		28C
				MTH MATRX NORM	147		
					156		
COL+	C	171-63	5.5	MTH MATRX COL	9		48GX
				MATRICES CREAT COL	149		
					165		
					2219.11		
COL-	C	171-62	5.5	MTH MATRX COL	9		48GX
				MATRICES CREAT COL	149		
					165		
					2219.11		
COLCT	C	2-333	2.5	93 MENU	93		28C
COLLECT	F	222-48	5.5	ALG	124	H	1.20
				«ALGB»			
COLΣ	C	2-312	2.5	CAT			28C
COL→	C	171-57	5.5	MTH MATRX COL	9		48GX
				MATRICES CREAT COL	149		
					165		
					2219.1		
COMB	F	2-129	2.5	MTH NXT PROB	13		28S
COMP→	C	256-13	5.5	256.03 MENU	256.03		1.05
CON	C	2-173	2.5	MATRICES CREAT	6		28C
				MTH MATRX MAKE	146		

Command	Type	Library	Size	Keys	Menu	?	First
RNRM	C	2-118	2.5	MTH MATRX NORM	7		28C
				MATRICES OPER NXT	147		
					156.02		
ROLL	C	2-280	2.5	PRG/TOOL STACK NXT	73.02		28C
ROLLD	C	2-281	2.5	PRG/TOOL STACK NXT	73.02		28C
ROMUPLOAD	C	171-111	5.5	CAT – do not use	2219.19		1.16
ROOT	C	2-251	2.5	RS&NUM.SLV ROOT	75		28C
ROT	C	2-274	2.5	PRG/TOOL STACK	73		28C
ROW+	C	171-61	5.5	MTH MATRX ROW	10		48GX
				MATRICES CREAT ROW	150		
					166		
					2219.11		
ROW-	C	171-60	5.5	MTH MATRX ROW	10		48GX
				MATRICES CREAT ROW	150		
					166		
					2219.11		
ROW→	C	171-55	5.5	MTH MATRX ROW	10		48GX
				MATRICES CREAT ROW	150		
					166		
					2219.1		
RPL>	C	2-393	2.5	CAT			1.05
RR	C	2-3	2.5	[MTH/CONVERT] BASE NXT BIT	17		28C
					170		
RRB	C	2-4	2.5	[MTH/CONVERT] BASE NXT BYTE	18		28C
					171		
RREF	C	171-52	5.5	MATRICES LIN-S	8		48GX
				MTH MATRX FACTR	158		
					2219.09		
RREFMOD	C	788-120	5.5	120 DUP MENUXY		H	1.05
RRK	C	171-35	5.5	RS&NUM.SLV DIFFEQ	76		48GX
					2219.06		
RRKSTEP	C	171-36	5.5	RS&NUM.SLV DIFFEQ	76		48GX
					2219.07		
RSBERR	C	171-37	5.5	RS&NUM.SLV DIFFEQ	76		48GX
					2219.07		
RSD	C	2-123	2.5	MTH MATRX NXT	5.02		28C
				RS&NUM.SLV SYS	78		
				MATRICES OPER NXT	156.02		
RSWP	C	171-64	5.5	MTH MATRX ROW NXT	10.02		48GX
				MATRICES CREAT ROW NXT	150.02		
					166.02		
					2219.11		
RULES	C	2-335	2.5	CAT			48SX
R~SB	C	256-19	5.5	256.04 MENU	256.04		1.05
R→B	C	2-9	2.5	[MTH/CONVERT] BASE	15		28C
					168		
R→C	C	2-153	2.5	PRG TYPE NXT	20		28C
				MTH NXT CMPLX	33.02		
R→D	F	2-113	2.5	MTH REAL NXT NXT	14.03		28C

Command	Type	Library	Size	Keys	Menu	?	First
DATE+	C	2-31	2.5	RS&TIME NXT	94.02		48SX
				PRG NXT NXT TIME NXT	167.02		
				APPS 5 4 NXT			
DBUG	C	221-21	5.5	CAT	41		1.10
					2269.04		
DDAYS	C	2-30	2.5	RS&TIME NXT	94.02		48SX
				PRG NXT NXT TIME NXT	167.02		
				APPS 5 4 NXT			
DEC	C	2-145	2.5	[MTH/CONVERT] BASE	15		28C
					168		
DECR	C	2-332	2.5	PRG MEM ARITH	72		48SX
DEDICACE	C	222-55	5.5	CAT			1.20
DEF	F	222-37	5.5	«ALGB»		H	1.20
DEFINE	C	2-343	2.5	key 93.2	153		48SX
				SYMB GRAPH	177		
				CALC GRAPH			
DEG	C	2-135	2.5	LS&MODE ANGLE	65		28C
				PRG NXT MODES ANGLE			
DEGREE	F	222-54	5.5	CAT		H	1.20
DELALARM	C	39872	2.5	RS&TIME ALRM	95		48SX
				PRG NXT NXT TIME ALRM			
DELAY	C	2-245	2.5	108 MENU	108		48SX
DELKEYS	C	2-382	2.5	LS&MODE KEYS	67		48SX
				PRG NXT MODES KEYS			
DEPND	C	2-196	2.5	83 MENU	83		48SX
DEPTH	C	2-276	2.5	PRG/TOOL STACK NXT	73.02		28C
DERIV	F	788-14	5.5	SYMB CALC	136	H	1.05
				CALC DERIV	152		
				«DIFF»	162		
DERVX	F	788-3	5.5	CALC	123	H	1.05
				SYMB CALC	136		
				CALC DERIV	152		
				«DIFF»	162		
DESOLVE	C	788-15	5.5	S.SLV	120	H	1.05
				CALC DIFF	136		
				«SOLVER»	164		
DET	F	2-120	2.5	MATRICES OPER	7.02		28C
				MTH MATRX NORM NXT	137		
					147		
					156		
DETACH	C	2-359	2.5	110 MENU	110		48SX
DIAGMAP	C	222-12	5.5	MATRICES NXT EIGEN	159	H	1.20
DIAG→	C	171-59	5.5	MATRICES CREAT NXT	5.02		48GX
				MTH MATRX NXT	6.03		
				MTH MATRX MAKE NXT NXT	146.02		
					2219.1		
DIFF	C	788-132	5.5	«MAIN»		H	1.05
DIFFEQ	C	171-14	5.5	82 MENU	82		48GX
					2219-03		

Command	Type	Library	Size	Keys	Menu	?	First
PVIEW	C	2-202	2.5	PRG NXT OUT PRG NXT PICT NXT	38.02 40		48SX
PWRFIT	C	2-322	2.5	90/99 MENU	90 99		48SX
PX→C	C	2-198	2.5	PRG NXT PICT NXT	38.02		48SX
Psi	F	222-3	5.5	MTH NXT SPECIAL		H	1.16
QR	C	171-49	5.5	MATRICES FACT MTH MATRX FACTR	8 148 2219.09		48GX
QUAD	C	2-337	2.5	93 MENU	93		28C
QUOT	F	788-40	5.5	ARITH POLY PREV «POLYNOMIAL» NXT	126.04	H	1.05
QUOTE	F	2-257	2.5	«ALGB» 93.03 MENU	93.03		48SX
QXA	C	788-75	5.5	MATRICES QUADF CONVERT MATRX «MATR» NXT	157 173	H	1.05
RAD	C	2-136	2.5	LS&MODE ANGLE PRG NXT MODES ANGLE	65		28C
RAND	C	2-127	2.5	MTH NXT PROB	13		28C
RANK	C	171-42	5.5	MATRICES OPER NXT MTH MATRX NORM NXT	7.02 147 156.02 2219.08		48GX
RANM	C	171-53	5.5	MTH MATRX MAKE MATRICES CREAT NXT NXT	6 146.03 2219.09		48GX
RATIO	F	2-268	2.5	CAT			48SX
RCEQ	C	2-249	2.5	RS&NUM.SLV ROOT RS-EQ	75		28C
RCI	C	171-66	5.5	MTH MATRX ROW MATRICES CREAT ROW	10 150 166 2219.12		48GX
RCIJ	C	171-67	5.5	MTH MATRX ROW MATRICES CREAT ROW	10 150 166 2219.12		48GX
RCL	C	2-340	2.5	key 32.2 TOOL PRG MEM DIR	71		28C
RCLALARM	C	2-26	2.5	RS&TIME ALRM PRG NXT NXT TIME ALRM	95		48SX
RCLF	C	2-150	2.5	LS&MODE FLAG NXT PRG NXT MODES FLAG NXT	66.02		28C
RCLKEYS	C	2-383	2.5	LS&MODE KEYS PRG NXT MODES KEYS	67		48SX
RCLMENU	C	2-350	2.5	LS&MODE MENU PRG NXT MODES MENU	68		48SX
RCLVX	C	222-63	5.5	CAT		H	1.20

Command	Type	Library	Size	Keys	Menu	?	First
EGCD	C	788-46	5.5	ARITH POLY «POLYNOMIAL»	126	H	1.05
EGV	C	171-44	5.5	MATRICES NXT EIGEN MTH MATRX NXT	5.02 137 159 2219.08		48GX
EGVL	C	171-45	5.5	MATRICES NXT EIGEN MTH MATRX NXT	5.02 137 159 2219.08		48GX
ELSE	C	1792-2	2.5	PRG BRCH IF PRG NXT NXT ERROR IFERR	24 60 145		28C
END	C	1792-23 1792-3 1792-22	2.5	PRG BRCH IF/CASE/DO/WHILE PRG NXT NXT ERROR IFERR	24 25 29 31 60 145		28C
ENDSUB	F	171-87	5.5	PRG LIST PROC	36 2219.15		48GX
ENG	C	2-140	2.5	LS&MODE FMT PRG NXT MODES FMT	64		28C
EPSX0	C	788-136	5.5	«REWRITE»		H	1.05
EQNLIB	C	227-0	5.5	APPS 12 EQLIB 114 MENU	2275		48GX
EQW	C	221-11	5.5	CAT (not the same as the EQW key) key 43.3	2269.02		1.05
EQ→	C	2-168	2.5	PRG TYPE NXT	33.02		48SX
ER	C	256-31	5.5	257 MENU 256.06 MENU	256.06		1.16
ERASE	C	2-197	2.5	81 MENU	81		48SX
ERR0	C	2-43	2.5	PRG NXT NXT ERROR	61		48SX
ERRM	C	2-45	2.5	PRG NXT NXT ERROR	61		28C
ERRN	C	2-44	2.5	PRG NXT NXT ERROR	61		28C
EULER	F	788-56	5.5	ARITH INTEG «INTEGER»	127	H	1.05
EVAL	C	2-46	2.5	key 41.1 in EQW in FILER 142 MENU	142		28C
EXLR	C	788-108	5.5	108 DUP MENUXY		H	1.05
EXP	A	2-93	2.5	key 51.2: e^x 141 MENU	141		28C
EXP&LN	C	788-135	5.5	«MAIN» NXT		H	1.05
EXP2HYP	F	222-62	5.5	CAT		H	1.20
EXP2POW	F	222-26	5.5	CONVERT REWRITE «REWRITE»	172	H	1.20

Command	Type	Library	Size	Keys	Menu	?	First
PARSURFACE	C	171-9	5.5	85 MENU	85 2219.02		48GX
PARTFRAC	C	788-52	5.5	ALG ARITH POLY NXT NXT «ALGB» «POLYNOMIAL» NXT	124 126.03	H	1.05
PATH	C	2-33	2.5	PRG MEM DIR	71		28S
PCAR	C	788-79	5.5	MATRICES NXT EIGEN «MATR» NXT	137.03 159	H	1.05
PCOEF	C	171-69	5.5	RS&NUM.SLV POLY ARITH POLY NXT NXT	77 126.03 2219.12		48GX
PCONTOUR	C	171-13	5.5	85 MENU	85 2219.03		48GX
PCOV	C	171-31	5.5	102.02 MENU	102.02 2219.06		48GX
PDIM	C	2-195	2.5	PRG NXT PICT	38		48SX
PEEK	C	256-17	5.5	256.03 MENU	256.03		1.05
PEEKARM	C	256-36	5.5	256.07 MENU	256.07		2.00
PERINFO	C	229-3	5.5	APPS 13			2.15
PERM	F	2-130	2.5	MTH NXT PROB	13		28S
PERTBL	C	229-0	5.5	APPS 13			2.15
PEVAL	C	171-70	5.5	RS&NUM.SLV POLY	77 2219.12		48GX
PGDIR	C	2-352	2.5	PRG MEM DIR	71		48SX
PICK	C	2-279	2.5	PRG/TOOL STACK NXT	73.02		28C
PICK3	C	2-397	2.5	PRG/TOOL STACK NXT	73.02		1.05
PICT	C	2-210	2.5	PRG NXT PICT	38		48SX
PICTURE	C	2-200	2.5	CAT key leftarrow when not editing			48GX
PINIT	C	171-106	5.5	110 MENU	110 2219.18		48GX
PIX?	C	2-205	2.5	PRG NXT PICT NXT	38.02		48SX
PIXOFF	C	2-204	2.5	PRG NXT PICT NXT	38.02		48SX
PIXON	C	2-203	2.5	PRG NXT PICT NXT	38.02		48SX
PKT	C	2-378	2.5	105 MENU	105		48SX
PLOT	C	788-9	5.5	SYMB GRAPH CALC GRAPH	153 177	H	1.05
PLOTADD	C	788-10	5.5	SYMB GRAPH CALC GRAPH	153 177	H	1.05
PMAX	C	2-185	2.5	CAT			28C
PMIN	C	2-184	2.5	CAT			28C
PMINI	C	222-20	5.5	MATRICES NXT EIGEN	159	H	1.20
POKE	C	256-16	5.5	256.03 MENU	256.03		1.05
POKEARM	C	256-35	5.5	256.06 MENU	256.06		2.00
POLAR	C	2-222	2.5	82 MENU	82		48SX
POLYNOMIAL	C	222-45	5.5	«ARIT» «MATHS»		H	1.20

Command	Type	Library	Size	Keys	Menu	?	First
FINDALARM	C	2-27	2.5	RS&TIME ALRM PRG NXT NXT TIME ALRM	95		48SX
FINISH	C	2-368	2.5	105 MENU	105		48SX
FIX	C	2-138	2.5	LS&MODE FMT PRG NXT MODES FMT	64		28C
FLASHEVAL	F	171-23	5.5	CAT	2219.04		1.05
FLOOR	F	2-103	2.5	MTH REAL NXT NXT «CMPLX»	14.03		28C
FONT6	C	221-15	5.5	CAT	2269.03		1.05
FONT7	C	221-14	5.5	CAT	2269.03		1.05
FONT8	C	221-13	5.5	CAT	2269.03		1.05
FONT→	C	221-3	5.5	CAT	2269		1.05
FOR	C	1792-10	2.5	PRG BRCH [FOR]	27 145		28C
FOURIER	F	788-94	5.5	CALC DERIV «DIFF»	136.02 162	H	1.05
FP	F	2-102	2.5	MTH REAL NXT	14.02		28C
FREE	C	2-356	2.5	CAT – do not use			48SX
FREEZE	C	2-51	2.5	PRG NXT OUT	40		48SX
FROOTS	C	788-66	5.5	ARITH POLY NXT «SOLVER»	126.02	H	1.05
FS?	C	2-133	2.5	LS&MODE FLAG PRG TEST NXT NXT PRG NXT MODES FLAG	32.03 66		28C
FS?C	C	2-142	2.5	LS&MODE FLAG PRG TEST NXT NXT PRG NXT MODES FLAG	32.03 66		28C
FUNCTION	C	2-220	2.5	82 MENU	82		48SX
FXND	C	788-107	5.5	107 DUP MENUXY		H	1.05
GAMMA	F	222-7	5.5	MTH NXT SPECIAL	176	H	1.16
GAUSS	C	788-77	5.5	MATRICES QUADF «MATR» NXT	157	H	1.05
GBASIS	C	222-58	5.5	CAT		H	1.20
GCD	F	788-44	5.5	ARITH POLY NXT «INTEGER» «POLYNOMIAL»	126.02	H	1.05
GCDMOD	F	788-117	5.5	ARITH MODUL «MODULAR»	128 138	H	1.05
GET	C	2-178	2.5	PRG LIST ELEM MATRICES CREAT NXT MTH MATRX MAKE NXT	6.02 35 146.02		28C
GETI	C	2-179	2.5	PRG LIST ELEM MATRICES CREAT NXT MTH MATRX MAKE NXT	6.02 35 146.02		28C
GOR	C	2-211	2.5	PRG NXT GROB	37		48SX
GRAD	C	2-137	2.5	LS&MODE ANGLE PRG NXT MODES ANGLE	65		48SX
GRAMSCHMIDT	C	222-9	5.5	MATRICES NXT VECT	160	H	1.20

Command	Type	Library	Size	Keys	Menu	?	First
MAD	C	788-81	5.5	MATRICES OPER NXT	137.02	H	1.05
				«MATR» NXT NXT	156.02		
MAIN	C	788-127	5.5	«ARIT», «CONSTANTS»		H	1.05
MAKESTR	C	256-28	5.5	256.05 MENU	256.05		1.16
MANT	F	2-111	2.5	MTH REAL NXT	14.02		28C
MAP	C	788-102	5.5	102 DUP MENUXY		H	1.05
MATHS	C	222-47	5.5	«MAIN»		H	1.20
MATR	C	788-131	5.5	«MAIN» NXT		H	1.05
MAX	F	2-106	2.5	MTH REAL	14		28C
MAXR	F	2-64	2.5	MTH NXT CONST NXT	21.02		28C
MAXΣ	C	2-296	2.5	100 MENU	100		28C
MCALC	C	171-118	5.5	APPS 12 MES	116		48GX
				116 MENU	2219.2		
MEAN	C	2-297	2.5	100 MENU	100		28C
MEM	C	2-345	2.5	PRG MEM	70		28C
MENU	C	2-349	2.5	LS&MODE MENU	68		28S
				PRG NXT MODES MENU			
MENUXY	C	788-122	5.5	788.21 MENU	788.21	H	1.05
MERGE	C	2-355	2.5	CAT – do not use			48SX
MIN	F	2-107	2.5	MTH REAL	14		28C
MINEHUNT	C	227-3	5.5	APPS 12 UTILS	2275		48GX
				117 MENU			
MINIFONT→	C	221-18	5.5	CAT	2269.04		1.05
MINIT	C	171-115	5.5	APPS 12 MES	116		48GX
				116 MENU	2219.2		
MINR	F	2-65	2.5	MTH NXT CONST NXT	21.02		28C
MINΣ	C	2-298	2.5	100 MENU	100		28C
MITM	C	171-116	5.5	APPS 12 MES	116		48GX
				116 MENU	2219.2		
MKISOM	C	222-14	5.5	MATRICES LINAP	175	H	1.20
MOD	F	2-110	2.5	MTH REAL	14		28C
				ARITH MODUL NXT	128.02		
				«CMPLX» NXT			
MODSTO	C	788-121	5.5	ARITH MODUL NXT	128.02	H	1.05
				«MODULAR» NXT	138.02		
MODULAR	C	222-44	5.5	«ARIT»		H	1.20
				«MATHS»			
MOLWT	F	229-2	5.5	APPS 13			2.15
MROOT	C	171-119	5.5	APPS 12 MES	116		48GX
				116 MENU	2219.2		
MSGBOX	C	171-78	5.5	PRG NXT OUT	40		48GX
					2219.14		
MSLV	C	222-32	5.5	NUM.SLV 6	132	H	1.20
				132 MENU			
MSOLVR	C	171-114	5.5	APPS 12 EQLIB/MES	116		48GX
				114/116 MENU	2219.2		
					2275		
MSOLVR2	C	227-2	5.5	CAT			2.00

Command	Type	Library	Size	Keys	Menu	?	First
IBASIS	C	222-18	5.5	MATRICES NXT VECT	160	H	1.20
IBERNOULLI	F	222-6	5.5	ARITH INTEG	127	H	1.16
IBP	C	788-11	5.5	SYMB CALC	136.02	H	1.05
				CALC DERIV NXT	152		
				«DIFF»	162.02		
ICHINREM	C	788-59	5.5	ARITH INTEG	127	H	1.05
IDIV2	C	788-39	5.5	ARITH INTEG	127	H	1.05
IDN	C	2-174	2.5	MATRICES CREAT	6		28C
				MTH MATRX MAKE	137.02		
					146		
IEGCD	C	788-47	5.5	SYMB ARITH	127	H	1.05
				ARITH INTEG	134		
				«INTEGER»			
IF	C	1792-0	2.5	PRG BRCH [IF]	24		28C
					145		
IFERR	C	1792-13	2.5	PRG NXT NXT ERROR [IFERR]	60		28C
IFFT	C	171-27	5.5	MTH NXT FFT	19		48GX
					2219.05		
IFT	C	2-48	2.5	PRG BRCH NXT	23.02		28C
					145.03		
IFTE	F	2-47	2.5	PRG BRCH NXT	23.02		28C
				«TESTS» NXT	145.03		
ILAP	F	788-17	5.5	CALC DIFF	136.02	H	1.05
				«DIFF» NXT NXT	164		
IM	F	2-155	2.5	CMPLX	20		28C
				MTH NXT CMPLX	130		
				«CMPLX» NXT	141		
IMAGE	C	222-16	5.5	MATRICES LINAP	175	H	1.20
INCR	C	2-331	2.5	PRG MEM ARITH	72		48SX
INDEP	C	2-183	2.5	83 MENU	83		28C
INFORM	C	171-76	5.5	PRG NXT IN	39		48GX
					2219.13		
INPUT	C	2-379	2.5	PRG NXT IN	39		48SX
INT	F	2-386	2.5	CAT			1.05
INTEGER	C	222-41	5.5	«ARIT»		H	1.20
				«MATHS»			
INTVX	F	788-4	5.5	CALC	123	H	1.05
				SYMB CALC	136.02		
				CALC DERIV NXT	152		
				«DIFF»	162.02		
INV	A	2-76	2.5	key 64.1: 1/x			28C
INVMOD	F	788-116	5.5	ARITH MODUL NXT	128.02	H	1.05
				«MODULAR»	138		
IP	F	2-101	2.5	MTH REAL NXT	14.02		28C
IQUOT	F	788-41	5.5	SYMB ARITH	127.02	H	1.05
				ARITH INTEG NXT	134		
				«INTEGER»			

Command	Type	Library	Size	Keys	Menu	?	First
ΣLINE	C	2-314	2.5	102 MENU	102		48SX
ΣLIST	C	171-89	5.5	MTH LIST	11 2219.15		48GX
ΣX	C	2-291	2.5	103 MENU	103		28C
ΣX^2	C	2-295	2.5	alias for ΣX2		-	28C
ΣX2	C	2-293	2.5	103 MENU	103		1.05
ΣX*Y	C	2-295	2.5	alias for ΣXY	103	-	28C
ΣXY	C	2-293	2.5	103 MENU			1.05
ΣY	C	2-292	2.5	103 MENU	103		28C
ΣY^2	C	2-294	2.5	alias for ΣY2	103	-	28C
ΣY2	C	2-294	2.5	103 MENU			1.05
▶	F	2-342	2.5	key 32.1: STO			1.05
π	F	2-63	2.5	key 104.2 MTH NXT CONST «CONSTANTS»	21		28C
∂	F	2-247 2-248	2.5	key 54.3			28C
≤	F	2-237	2.5	key 63.2 PRG TEST «TESTS»	32		28C
≥	F	2-238	2.5	key 64.2 PRG TEST «TESTS»	32		28C
≠	F	2-234	2.5	key 62.2 PRG TEST «TESTS» NXT	32		28C
→	C	1792-4 1792-16	2.5	key 102.3			28C
→A	C	256-2	5.5	256 MENU	256		1.05
→ALG	C	256-11	5.5	256.02 MENU	256.02		1.05
→ARRY	C	2-170	2.5	PRG TYPE	33		28C
→CD	C	256-6	5.5	256.02 MENU	256.02		1.05
→COL	C	171-56	5.5	MTH MATRX COL MATRICES CREAT COL	9 149 165 2219.1		48GX
→DATE	C	2-22	2.5	RS&TIME PRG NXT NXT TIME APPS 5 4	94 167		48SX
→DIAG	C	171-58	5.5	MATRICES CREAT MTH MATRX NXT MTH MATRX MAKE NXT NXT	5.02 6.03 146 2219.1		48GX
→FONT	C	221-2	5.5	CAT	2269		1.05
→GROB	C	2-215	2.5	PRG NXT GROB	37		48SX
→H	C	256-0	5.5	256 MENU	256		1.05
→HEADER	C	221-4	5.5	CAT	2269		1.05

ASCII Character Codes and Translations

The following table shows the relation between character codes (results of NUM, arguments to CHR) and characters (results of CHR, arguments to NUM) for the lower half of the character set.

Character Codes (0-127)

Code		Description
0	▪	null
1	▪	start of heading
2	▪	start of text
3	▪	end of text
4	▪	end of transmission
5	▪	enquiry
6	▪	acknowledge
7	▪	bell
8	▪	backspace
9	▪	tab
10	↵	linefeed
11	▪	vertical tab
12	▪	form feed
13	▪	carriage return
14	▪	shift out
15	▪	shift in
16	▪	device escape
17	▪	device control 1
18	▪	device control 2
19	▪	device control 3
20	▪	device control 4
21	▪	neg. acknowledge
22	▪	synchronous idle
23	▪	end trans. block
24	▪	cancel
25	▪	end of medium
26	▪	substitute
27	▪	escape
28	↞	cursor (insert)
29	▓	cursor (overtype)
30	…	ellipsis (left)
31	…	ellipsis (right)
32		space
33	!	exclamation mark
34	"	quotation
35	#	number
36	$	dollars
37	%	percent
38	&	ampersand
39	'	apostrophe
40	(	left parenthesis
41	)	right parenthesis
42	*	asterisk

Code		Description
43	+	plus
44	,	comma
45	-	hyphen
46	.	period
47	/	forward slash
48	0	0
49	1	1
50	2	2
51	3	3
52	4	4
53	5	5
54	6	6
55	7	7
56	8	8
57	9	9
58	:	colon
59	;	semicolon
60	<	less than
61	=	equal to
62	>	greater than
63	?	question mark
64	@	at sign
65	A	A
66	B	B
67	C	C
68	D	D
69	E	E
70	F	F
71	G	G
72	H	H
73	I	I
74	J	J
75	K	K
76	L	L
77	M	M
78	N	N
79	O	O
80	P	P
81	Q	Q
82	R	R
83	S	S
84	T	T
85	U	U

Code		Description
86	V	V
87	W	W
88	X	X
89	Y	Y
90	Z	Z
91	[	left bracket
92	\	backslash
93	]	right bracket
94	^	caret
95	_	underscore
96	`	grave accent
97	a	a
98	b	b
99	c	c
100	d	d
101	e	e
102	f	f
103	g	g
104	h	h
105	i	i
106	j	j
107	k	k
108	l	l
109	m	m
110	n	n
111	o	o
112	p	p
113	q	q
114	r	r
115	s	s
116	t	t
117	u	u
118	v	v
119	w	w
120	x	x
121	y	y
122	z	z
123	{	left brace
124	|	pipe
125	}	right brace
126	~	tilde
127	▒	shaded box

Index

F