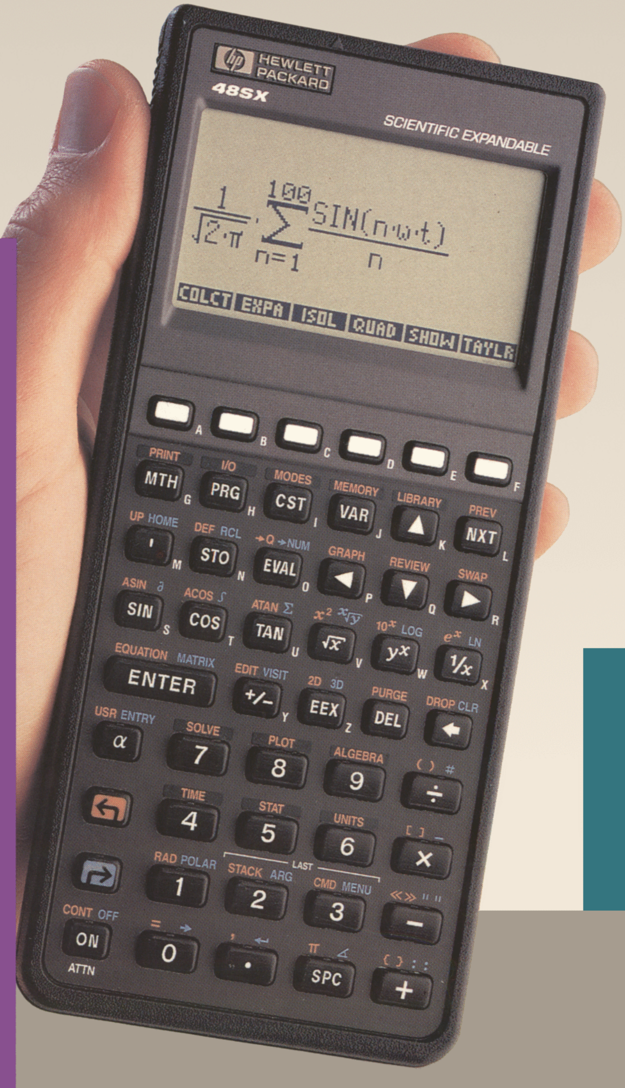
HEWLETT PACKARD

HP 48
Owner's Manual



Regulatory Information

U.S.A.

The HP 48 generates and uses radio frequency energy and may interfere with radio and television reception. The HP 48 complies with the limits for a Class B computing device as specified in Part 15 of FCC Rules, which provide reasonable protection against such interference in a residential installation. In the unlikely event that there is interference to radio or television reception (which can be determined by turning the unit off and on), try the following:

- Reorienting or relocating the receiving antenna.
- Relocating the HP 48 with respect to the receiver.

For more information, consult your dealer, an experienced radio/television technician, or the following booklet, prepared by the Federal Communications Commission: *How to Identify and Resolve Radio-TV Interference Problems.* This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock Number 004-000-00345-4. At the first printing of this manual, the telephone number was (202) 783-3238.

Europe

Declaration of Conformity (according to EN 45014)

Manufacturer's name:	Hewlett-Packard	
Manufacturer's address:	Corvallis Division	Singapore Mfg. Div.
	1000 NE Circle Blvd.	1150 Depot Road
	Corvallis, OR 97330	Singapore 0410
	declares that the following product	
Product name:	HP 48	
	conforms to the following product specifications	
EMC:	CISPR 22 / EN 55022 class B,	
	prEN 55101-2, prEN55101-3	
Safety:	IEC 950 / EN 60950	

Quality Department
Hewlett-Packard Company
Corvallis Division

Comments on the HP 48 Owner's Manual

Your evaluation of this manual helps us improve our publications. Please *circle* a response for each of the statements below.

HP 48 Owner's Manual

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
①	②	③	④	⑤

I am satisfied with the product documentation	1	2	3	4	5
I can find the information I want	1	2	3	4	5
The information in the manual is accurate	1	2	3	4	5
I can easily understand the information	1	2	3	4	5
The manual contains enough examples	1	2	3	4	5
The examples are appropriate and helpful	1	2	3	4	5

Comments: ______________________________

____

____

____

____

____

Name: ______________________________

Address: ______________________________

City/State/Zip: ______________________________

Occupation: ______________________________

Phone: (______) ______________________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 38 CORVALLIS, OR

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD COMPANY
DOCUMENTATION DEPARTMENT
CORVALLIS DIVISION
1000 NE CIRCLE BLVD
CORVALLIS OR 97330-9973

HP 48 Owner's Manual

The contents of this manual are printed on recycled paper.

HEWLETT PACKARD

HP Part No. 00048-90091
Printed in U.S.A. November 1991

Edition 1

Notice

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

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

Printing History

Edition 1 .... November 1991

Contents

Part 1. Building Blocks

Part 2. Hand Tools

9. Common Math Functions

10. User-Defined Functions

11. Complex Numbers

21. Statistics

22. Algebra

Part 1

Building Blocks

1

Trying Out the HP 48

This first chapter gives you a chance to try a few things with your HP 48. You'll see how to do a variety of basic tasks—using some of the tools provided by the HP 48.

If you're already familiar with using the HP 41 or a similar RPN calculator, you may want to skip ahead to appendix F, "Comparing the HP 48 and HP 41."

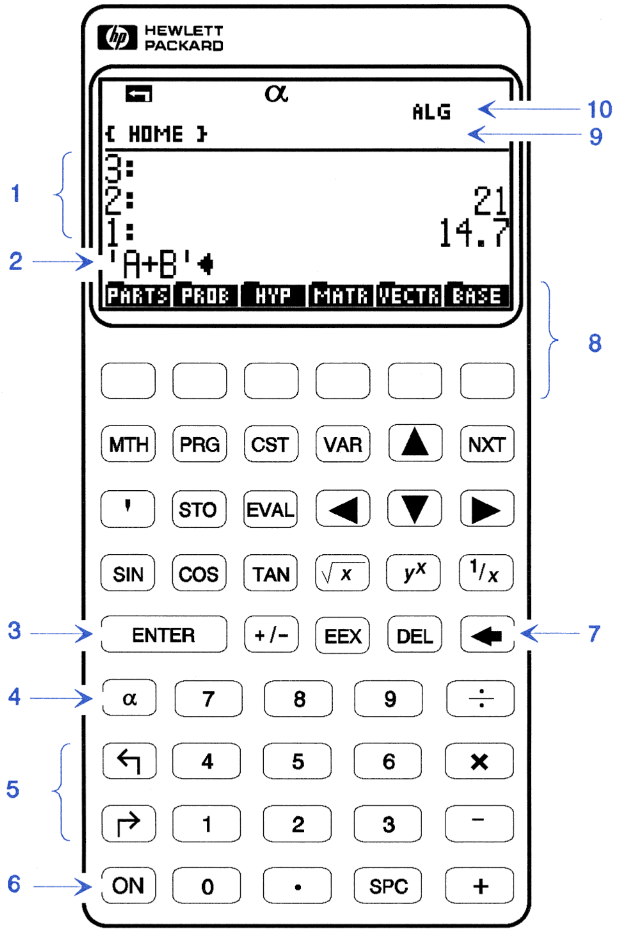
Here are some suggestions for using this chapter:

- Try the examples. They'll give you a good idea of how you can use the HP 48.
- Headings highlight the steps for doing certain tasks. You can experiment, if you want—but the examples are probably your best guide for trying things out.
- You can turn off the calculator at any time—when you turn it on again, it'll be ready to continue where you left off.
- If you run into trouble, see "If Things Go Wrong" on page A-1.

Note This chapter gives only selected information about the HP 48. See the other chapters to get more complete information.

---

1 Looking It Over



1. Stack levels
2. Command line
3. Enter command line
4. Alpha mode
5. Shift keys
6. On, off, attention
7. Backspace
8. Menu labels and keys
9. Message area
10. Annunciators

1. **Stack levels.** The stack holds the data you're currently working with. Each numbered stack level holds one item of data, called an *object*. The stack can have more levels than show in the display.
2. **Command line.** Numbers and other text you type accumulate in the command line. The command line is displayed only while you're using it.
3. **Enter command line.** The [ENTER] key processes the text in the command line.
4. **Alpha mode.** The [α] key turns the alpha keyboard on and off. You use the alpha keyboard to type letters and other characters.
5. **Shift keys.** The orange [◀] (left-shift) key activates the operations labeled in orange above most keys. The blue [▶] (right-shift) key activates the operations labeled in blue, plus other unlabeled operations.
6. **On, off, attention.** The [ON] key turns on the HP 48. After it's on, the key means "Attention!"—it deletes the command line and stops whatever's going on. You also use [▶] and this key to turn off the HP 48.
7. **Backspace.** If there's a command line, [◆] deletes the character to the left of the cursor. If there's no command line, this key deletes the contents of stack level 1.
8. **Menu labels and keys.** The labels at the bottom of the display show the operations for the six menu keys below.
9. **Message area.** This part of the display shows the name of the current directory and any prompts and messages.
10. **Annunciators.** Annunciators indicate the current status of the calculator, including shift-key and alpha status.

The keyboard contains labels for many operations. Some are printed on the keys themselves—the "main" operations. Others are printed in orange and blue above the keys—the "shifted" operations.

The following table lists the labels for the main and shifted operations on the keyboard—plus it shows the row and position where each label occurs, counted from the top and left. You may find this helpful while you're becoming familiar with the HP 48 keyboard—especially as you try the examples in this manual.

Keys and Their Row/Position Locations

ACOS	4,2	LAST STACK	8,3	STAT	7,3	α	6,1
ALGEBRA	6,4	LIBRARY	2,5	STO	3,2	π	9,4
ASIN	4,1	LN	4,6	SWAP	3,6	Σ	4,3
ATAN	4,3	LOG	4,5	TAN	4,3	∂	4,1
ATTN	9,1	MATRIX	5,1	TIME	7,2	$\int$	4,2
CLR	5,5	MEMORY	2,4	UNITS	7,4	$\measuredangle$	9,4
CONT	9,1	MODES	2,3	UP	3,1	⬅	5,5
COS	4,2	MTH	2,1	USR	6,1	$\rightarrow$	9,2
CST	2,3	→NUM	3,3	VAR	2,4	$\hookleftarrow$	9,3
DEF	3,2	NXT	2,6	VISIT	5,2	'	3,1
DEL	5,4	OFF	9,1	x^2	4,4	.	9,3
DROP	5,5	ON	9,1	$\sqrt{x}$	4,4	,	9,3
EDIT	5,2	PLOT	6,3	$\sqrt[x]{y}$	4,4	#	6,5
EEX	5,3	POLAR	8,2	y^x	4,5	_	7,5
ENTER	5,1	PREV	2,6			( )	6,5
ENTRY	6,1	PRG	2,2	$1/x$	4,6	[ ]	7,5
EQUATION	5,1	PRINT	2,1	10^x	4,5	« »	8,5
EVAL	3,3	PURGE	5,4	2D	5,3	" "	8,5
e^x	4,6	→Q	3,3	3D	5,3	{ }	9,5
GRAPH	3,4	RAD	8,2	+/−	5,2	: :	9,5
HOME	3,1	RCL	3,2	+	9,5	▲	2,5
I/O	2,2	REVIEW	3,5	−	8,5	▼	3,5
LAST ARG	8,3	SIN	4,1	×	7,5	◀	3,4
LAST CMD	8,4	SOLVE	6,2	÷	6,5	▶	3,6
LAST MENU	8,4	SPC	9,4	=	9,2		

Getting Ready

To turn the HP 48 on and off:

- To turn it on, press [ON].
- To turn it off, press [↱][OFF]—that is, press the blue [↱] key, then press the key with the blue "OFF" label above it (the [ON] key).

To adjust the display contrast:

- To darken the display, turn on the HP 48, then hold down the [ON] key and press [+].
- To lighten the display, turn on the HP 48, then hold down the [ON] key and press [−].

The examples in this chapter assume the HP 48 is in its initial, default condition—they assume you haven't changed any of the HP 48 operating modes. (To reset the calculator to this condition, see "If Things Go Wrong" on page A-1.)

Operating the Calculator

When you use the HP 48, you use *commands* to process numbers or other *objects* and get results. Most of this action takes place in the *command line* and on the *stack*.

Some commands are labeled on the keyboard—others are presented in *menus* in the display. Your data might be numeric—or it might contain algebraic variables or text.

General rule for executing commands:

1. Enter the arguments for the command, if any.
2. Execute the command.

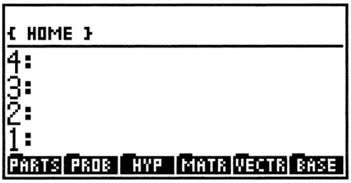
An *argument* is an item of data—an object—that's used by a command to get a result. The number of arguments depends on the command—some commands use no arguments, others use one or two or more. For example, "addition" requires two arguments, "tangent" requires one, and "set standard display" requires none.

The idea of entering two numbers and *then* executing "addition"—or entering an angle and *then* executing "tangent"—may seem unusual at first. But it's part of a consistent and efficient operating scheme that uses a stack-based syntax, sometimes called *RPN* ("Reverse Polish Notation"). (This is somewhat different from earlier HP RPN calculators, such as the HP 41, which prompted for information *after* certain commands, such as STO and FIX.)

To enter more than one argument, you can press [ENTER] after each argument—or you can press [SPC] to include more than one argument in the command line.

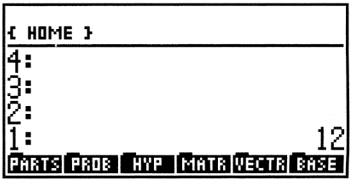
1 **Example:** Turn on the HP 48, then add 12 and 34.

[ON]



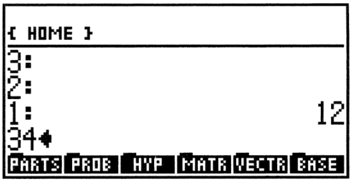
Type and terminate the first number.

12 [ENTER]



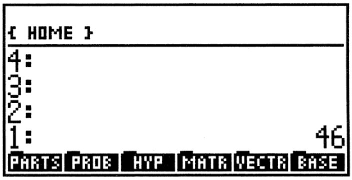
Type the second number.

34



Add the two numbers you've entered.

[+]



When you start to key in numbers or other data, the *command line* automatically appears near the bottom of the display—and your input shows up there.

When you execute a command—such as [ENTER] or [+] in the previous example—the command line disappears and the result shows up on the *stack*. The stack is a sequence of storage locations in memory, and the first few are shown in the display:

- As you enter numbers or other objects, previous objects move up to higher levels of the stack.

- As you delete objects from the bottom of the stack, the remaining objects move down.
- As you execute commands, they remove objects from the bottom of the stack and replace them with the results.
- The number of existing stack levels changes according to the number of objects present—from 0 to hundreds or more.

To delete the command line:

- Press ATTN. (That's the name of the ON key while the calculator is turned on.)

To delete the object in level 1 of the stack:

- Press ◆ or ↰ DROP, but only if there's no command line. (DROP is above the ◆ key.)

To clear the whole stack:

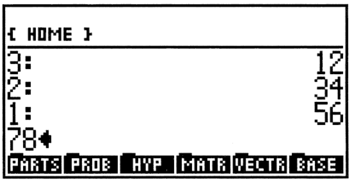
- Press ↱ CLR. (CLR is above the ◆ key.)

Objects are kept on the stack until you use or delete them. It's a good idea to clear the stack occasionally to recover memory.

Example: Enter four numbers and add the last two, then delete the result and clear the stack.

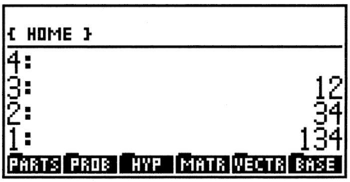
Clear the stack and enter 12, 34, 56, and 78.

↱ CLR
12 ENTER 34 ENTER 56 ENTER 78



Add the last two numbers.

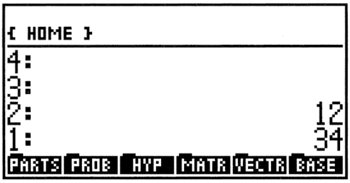
+



1

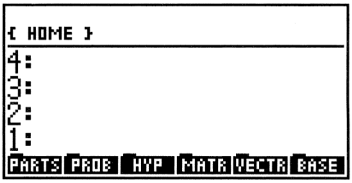
Delete the result.

[⇐]



Clear the whole stack.

[↱] [CLR]



To correct what you're typing in the command line:

1. Start typing a number or other object.
2. If necessary, press [◀] or [▶] to move the ✦ cursor to the error. (You can use [▲] and [▼] if the command line has more than one line—if the command line disappears when you press [▲] or [▼], press [ATTN] to recover.)
3. Delete the error:
   - To delete the character to the left of the cursor, press [⇐].
   - To delete the character under the cursor, press [DEL].
4. Type the correct characters.

To edit the object in level 1 of the stack:

1. Press [↰] [EDIT].
2. Edit the displayed information. Press [◀] or [▶] to move the cursor. Press [DEL] or [⇐] to delete characters. (You can use [▲] or [▼] if the object has more than one line.)
3. Press [ENTER] to save the changes (or press [ATTN] to discard them).

Example: Enter the number 5045.661, then change the "04" to "40".

Enter the number.

5045.661 [ENTER]

Start editing the number.

[←] [EDIT]

Make the changes.

[▶] [DEL] [DEL] 40

Save the changes.

[ENTER]

To get back to the stack display at any time:

- Press [ATTN]. In certain situations, you may have to press it more than once.

To view the whole stack:

1. Press [▲] while the stack is displayed and no command line is present.
2. To see other levels of the stack, press [▲] and [▼].
3. Press [ATTN] to return to normal operation.

Viewing the stack this way doesn't change the contents of the stack in any way.

To use a menu command:

1. Press the key or keys that get the menu you want.
2. If necessary, press [NXT] or [←] [PREV] to get to the menu page for the command you want.
3. If the command requires data, enter the data. (You can do this *before* you get the menu, if you want.)
4. Press the white menu key below the label for the command. (In this manual, normal menu labels and menu keys are shown like THIS.)

Many HP 48 commands are contained in *menus*—groups of operations labeled across the bottom of the display. The MTH, PRG, CST, and VAR keys get certain menus. In addition, shifted keys with orange labels on darker backgrounds get other menus, such as ← MODES.

Some menus contain other submenus—such as the MTH (math) menu. If a menu label has a "bar" over the top-left corner, it gets a submenu. For example, in the MTH menu, PARTS gets a submenu. Some menus contain more than six entries—so those menus have more than one "page" of labels. For example, the MTH PARTS menu has four pages—press NXT to see each page.

To go from a submenu to any other menu, just go to the new menu directly—you don't go back "up" from a submenu.

Example: Find 15 percent of 145. The % command is in the MTH PARTS menu.

Clear the stack and enter 145 and 15. Then get the MTH menu.

→ CLR
145 ENTER 15
MTH

Get the PARTS submenu, find the % command, and calculate 15% of 145.

PARTS NXT %

Example: Find 6! (6 factorial). The ! command is in the MTH PROB menu.

Clear the stack and key in 6. Then get the MTH PROB menu.

→ CLR
6 MTH PROB

Execute the ! command to find 6!.

!

To type letters and other characters:

- To type an individual letter, press (α), then press the key with that letter next to it.
- To type a sequence of letters, press (α) (α), press the sequence of letter keys, then press (α). (If you press (ENTER) to immediately enter the command line, you don't need the final (α).)
- To type any number of letters, hold down (α), press the letter keys, then release (α).

The α annunciator at the top of the display turns on while the "alpha" keyboard is active. If you press (α) twice, the α locks on until you press (α) again or process the command line.

Letter keys are labeled in white to the right of the keys. When α is on, the letters are active. You can type numbers with α on or off.

You can type commands and other kinds of information letter-by-letter, as you'll see throughout this chapter. In the examples in this manual, characters you type are shown only as "ABC"—but to type them, you have to use (α), such as

(α) A (α) B (α) C → ABC
(α) (α) A B C (α) → ABC
(α) (hold) A B C (release) → ABC

To enter any type of object:

1. Type the *delimiters* for the type of object you're entering, if any.
2. Enter the information by pressing command keys or typing characters.
3. Press (ENTER).

Each type of *object* represents a different kind of information. Here's a partial list of different types of objects and their corresponding *delimiters*—the punctuation that defines the type of object.

Objects	Delimiters	Examples
Real Number	none	`14.75`
Complex Number	`( )` (parentheses)	`(8.25,12.1)`
String	`" "` (quotes)	`"Hello"`
Array	`[ ]` (brackets)	`[ 4.8 -1.3 2.1 ]`
Unit	`_` (underscore)	`11.5_ft`
Program	`« »` (program quotes)	`« √ DUP NEG »`
Algebraic	`' '` (tick marks)	`'A-B'`
List	`{ }` (braces)	`{ 6.8 5 "FIVE" }`
Built-In Command	none	`FIX`
Name	none *or* `' '` (tick marks)	`VOL` *or* `'VOL'`

Most commands work with several types of objects—so you have fewer commands to remember. For example, you can add more than just numbers—the + command works with real numbers, complex numbers, arrays, strings, algebraic objects, and others.

Example: Enter the two text strings "HELLO" and " WORLD", then combine them.

Clear the stack and enter the first string. (Remember: Use [α] when you type the letters below.)

[↱] [CLR]
[↱] [" "] HELLO [ENTER]

Enter the second string—notice it starts with a space. (Use [α] when you type the letters below.)

[↱] [" "] [SPC] WORLD [ENTER]

Combine (add) the strings.

[+]

Using Memory

Although the stack can contain many pieces of information—many objects—the most convenient place to store information for later use is in *variables*. A variable is just a place in memory where an object is stored—*any* type of object. So a variable can contain a single number—or it can contain a complex program.

Each variable has a name you give it. You use the name to identify and access the object stored there.

(A variable is similar to a register in earlier HP calculators, such as the HP 41—except that a variable has a name you give it and can contain *any* type of object, even a program.)

To store any type of object in a new variable:

1. Enter the object.
2. Press ['] and type a name for the variable. (Use the [α] key as required—make sure the α annunciator is off when you're done.)
3. Press [STO].

You can use descriptive names for variables. A name can be as short as one letter—or as long as 127 characters. Names can't be the same as built-in commands and can't start with numbers.

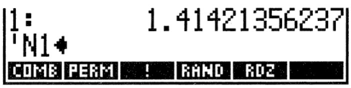
Example: Find the square root of 2 and store the value in a variable named *N1*.

Clear the stack and find the square root of 2.

[→] [CLR]
2 [√x]

Key in the name *N1*. (Use [α] when you type the letter below. Make sure the α annunciator is turned off at the end of this step.)

['] N1



Store the result.

[STO]

1

To store any type of object in an existing variable:

1. Enter the object.
2. Press [VAR].
3. Press [⬅] name for the name of the variable you want. (Substitute the desired menu key for name.)

The VAR (variable) menu contains names of existing variables.

To recall a stored object:

- Press [VAR], then press [➡] name for the name of the variable.

  or
- Press ['] and type the name of the variable, then press [➡][RCL].

To "use" a stored object:

- Press [VAR], then press name for the name of the variable.

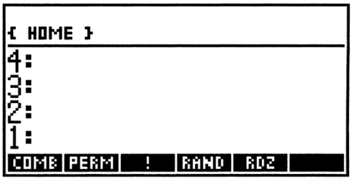
  or
- Key in the name of the variable (without ['] tick marks) and press [ENTER].

For numbers and other data-type objects, "using" an object simply means recalling its contents. For a program, "using" means running the program.

Example: Store the width and length of a 3-by-5 rectangle in *W* and *L*, then use those values to find the area. Store the result in existing variable *N1*.

Enter and store the width and length. (Use [α] when you type the letters below.)

3 [ENTER]
['] L [STO]
5 [ENTER]
['] W [STO]



Recall the two values.

[VAR] L W

Multiply to find the area.

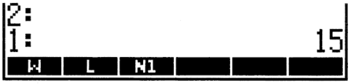
[×]

Store the area in existing variable *N1*.

[⬅] N1

Recall the area from *N1*.

N1



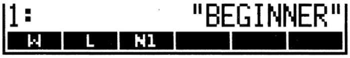
To edit ("visit") a stored object:

1. Press ['].
2. Press [VAR], then press name for the name of the variable.
3. Press [➡][VISIT].
4. Edit the displayed information. Press [◀], [▶], [▲], and [▼] to move the cursor. Press [DEL] or [⬅] to delete characters.
5. Press [ENTER] to save the changes (or press [ATTN] to discard them).

Example: Store the text string "BEGINNER" under the name *TXT*. Then change it to "WINNER".

Clear the stack and enter the text string in quotes. (Use [α] when you type the letters below.)

[➡][CLR]
[➡][" "] BEGINNER [ENTER]



Store the text in *TXT*. (Use [α] when you type the letters below.)

['] TXT [STO]

Get ready to edit the stored object. (Press [VAR] if you don't see the TXT menu label.)

['] TXT
[➡][VISIT]

Move the cursor to the first letter, then delete the first three letters and insert the new one. (Use [α] when you type the letter below.)

[▶] [DEL] [DEL] [DEL] W

Save the changes.

[ENTER]

To delete a stored object (the variable):

1. Press ['].
2. Press [VAR], then press name for the name of the variable.
3. Press [↰] [PURGE].

Example: Delete the text string stored in variable *TXT*.

Enter the name of the variable. (Press [VAR] if you don't see the *TXT* menu label.)

['] TXT

Delete the object—and the name disappears from the menu.

[↰] [PURGE]

To delete all stored objects (all variables):

1. *Don't do this if there are any stored objects you want to keep.*
2. Press [↱] [PURGE] (*right* shift).
3. Press [ENTER] to delete them (or press [ATTN] to *not* delete them.)

Ways to Solve Problems

You can use the HP 48 to solve problems in different ways. The next several topics introduce some of them:

- Doing numeric keyboard calculations (page 1-17).
- Doing algebra (page 1-22).
- Solving equations for unknown values (page 1-29).
- Getting answers graphically (page 1-33).
- Making your own functions (page 1-39).
- Programming (page 1-40).

Doing Numeric Keyboard Calculations

Numeric keyboard calculations are handy for one-time problems involving strictly numeric results. You can do *stack*-based calculations (described on the next few pages) and *algebraic* calculations (described after that). (These calculations also illustrate the basic syntax for other HP 48 commands.)

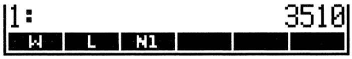
See "Using Memory" on page 1-13 to find out how to store and recall objects, such as numbers.

To calculate with two numbers:

1. Enter the first number.
2. Press ENTER (or SPC).
3. Enter the second number.
4. Press the command key.

Example: Calculate 45×78.

45 ENTER 78 ×



Example: Calculate 20^{-2}.

20 SPC 2 +/- y^x

Example: Calculate the percent change from 88 to 99.

88 ENTER 99
MTH PARTS NXT %CH

To calculate with one number:

1. Enter the number.
2. Press the command key.

Example: Calculate $\frac{1}{62.5}$.

62.5 $1/x$

Example: Calculate $\sqrt{166}$.

166 $\sqrt{x}$

To use previous results (with no command line):

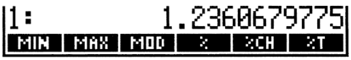
- To use the result in level 1 with a one-number calculation, press the command key.
- To use the result in level 1 with a two-number calculation, enter the second number, then press the command key.
- To use the results in levels 1 and 2 with a two-number calculation, press the command key.
- To swap the numbers in levels 1 and 2, press ▶ or ↰ SWAP.

Example: Calculate $12 + 13 + 14$.

12 ENTER 13 + 14 +

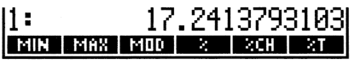
Example: Calculate $\sqrt{5} - 1$.

5 $\sqrt{x}$ 1 −



Example: Calculate $\frac{15}{.06 \times 14.5}$. (Pressing ▶ swaps the numerator into level 2 before pressing ÷.)

.06 ENTER 14.5 ×
15 ENTER ▶ ÷



To key in large and small numbers (powers of 10):

1. Key in the mantissa. Press [+/-] if the mantissa is negative.
2. Press [EEX]. (It types an E for "exponent.")
3. Key in the exponent—the power of 10. Press [+/-] if the exponent is negative.

For a number like -1.602×10^{-19}, the *mantissa* is -1.602 and the *exponent* is -19.

Example: Find the number of molecules in 13.5 grams of sodium hydroxide (NaOH). The solution is $N_A \times m / MW$, where N_A is Avogadro's number (6.022×10^{23}), m is 13.5, and MW is the sum of the atomic weights of sodium, oxygen, and hydrogen (23, 16, and 1).

Clear the stack, then enter Avogadro's number.

[⮫] [CLR]
6.022 [EEX] 23 [ENTER]

Multiply by 13.5.

13.5 [×]

Add the first two atomic weights.

23 [ENTER] 16 [+]

Add the third atomic weight to the previous result.

1 [+]

Divide the two values already on the stack.

[÷]

To enter the value of π:

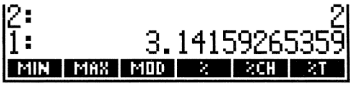
1. Press [↰] [π].
2. Press [↱] [→NUM].

π is normally expressed as a symbol—you have to press [↱] [→NUM] to get a numeric value.

Example: Calculate the value of 2π.

Clear the stack, enter 2, and enter the value of π. (You don't have to press [ENTER].)

[↱] [CLR]
2
[↰] [π] [↱] [→NUM]



Multiply the two numbers.

[×]

To calculate "algebraically":

1. Press ['].
2. Enter the numbers, operators, and parentheses in left-to-right order. Press [▶] to skip past right parentheses.
3. Press [EVAL] (or press [↱] [→NUM] if the expression contains π or other symbolic constant).

Example: Calculate 20^{-2}.

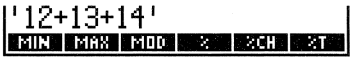
['] 20 [y^x] [−] 2

Evaluate the expression.

[EVAL]

Example: Calculate $12 + 13 + 14$.

['] 12 [+] 13 [+] 14



Evaluate the expression.

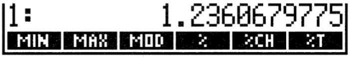
[EVAL]

Example: Calculate $\sqrt{5}-1$.

['] [√x] 5 [−] 1

Evaluate the expression.

[EVAL]



Example: Calculate $\frac{15}{.06 \times 14.5}$.

['] 15 [÷] [◀] [( )] .06 [×] 14.5

Evaluate the expression.

[EVAL]

To set the angular units for trig functions:

- To switch from degrees to radians, press [◀] [RAD].
- To switch back to degrees, press [◀] [RAD].

At the top of the display, RAD shows that radians are active—no annunciator means degrees are active. If you're a calculus student, you may want to have radians active.

To change the display format for numbers:

- For "standard" format, press [◀] [MODES] STD .
- For n decimal places, enter the number n, then press [◀] [MODES] FIX .
- For scientific format, enter the number of decimal places, then press [◀] [MODES] SCI .
- For engineering format, enter the number of digits after the first one, then press [◀] [MODES] ENG .
- To change the fraction mark to period (.) or to comma (,), press [◀] [MODES] [◀] [PREV] FM, .

In "standard" format, all numbers are shown with full precision—all significant digits after the decimal are displayed. Internally, full precision is always maintained, regardless of the display format.

Example: Calculate e^5. Then display it with 2 decimal places, in scientific format with 4 decimal digits, and with full precision.

Clear the stack and find e^5.

(⮫)(CLR)
5 (⮪)(e^x)

```
1:        148.413159103
 MIN  MAX  MOD   %   %CH  %T
```

Change to 2 Fix format.

2 (⮪)(MODES) FIX

```
1:               148.41
 STD  FIX■ SCI  ENG  SYM■ BEEP■
```

Change to 4 Sci format.

4 SCI

```
1:             1.4841E2
 STD  FIX  SCI■ ENG  SYM■ BEEP■
```

Change to Standard format.

STD

```
1:        148.413159103
 STD■ FIX  SCI  ENG  SYM■ BEEP■
```

Doing Algebra

You can do *symbolic math* on the HP 48—meaning you can calculate with symbols, as you do in algebra. So you can write equations on the HP 48, you can solve equations for certain variables, and you can "plug in" values and get numeric results.

Algebraic notation that contains an = sign is called an *equation*. Notation that does *not* contain an = sign is called an *expression*. For example, $x^2 + y^2 = r^2$ is an equation, and $1 + x^2$ is an expression.

You use *algebraic objects* to represent expressions and equations. You create them as described below. You store and recall them the same way you store and recall numbers—see "Using Memory" on page 1-13.

To enter an expression or equation using the EquationWriter application: 1

1. Press ⬅ EQUATION.
2. Key in the numbers, variables, operators, and parentheses in the expression or equation.
   - To key in a fraction, press ▲ to start the numerator. Press ▶ to end the numerator—and again to end the denominator.
   - To end each subexpression of an operator, press ▶. (See the explanation below.)
   - To correct a nearby typing mistake, press ◆ one or more times. (The ⧗ busy annunciator may turn on.)
   - To recover from a major mistake, press ATTN and start over.
3. Press ENTER (or press ATTN to discard the entry).

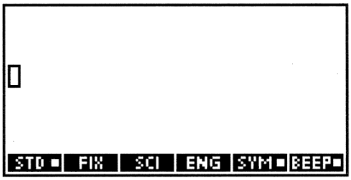
In the EquationWriter application, the arguments of algebraic functions are called *subexpressions*. Most subexpressions appear inside parentheses—some are displayed graphically, such as the subexpression for "square root." You press ▶ to end a subexpression and continue with the rest of the expression or equation.

Example: Use the EquationWriter application to enter this stress-analysis equation

$$\sin(2a) = \frac{t}{\sqrt{\left(\frac{sx-sy}{2}\right)^2 + t^2}}$$

Clear the stack and start the EquationWriter application. (If you make a mistake entering the equation, see step 2 above.)

➡ CLR
⬅ EQUATION



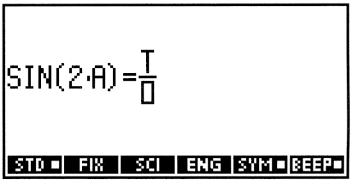
1

Key in the left side of the equation. (Use [α] when you type the letters in the equation.)

[SIN] 2A [▶]

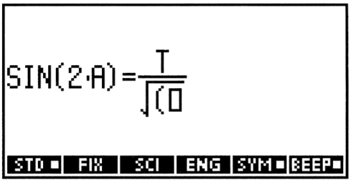
Key in the equal sign and the numerator of the fraction.

[⬑] [=]
[▲] T [▶]



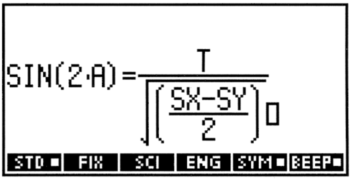
Key in the square root and start the parenthetic expression.

[√x]
[⬑] [()]



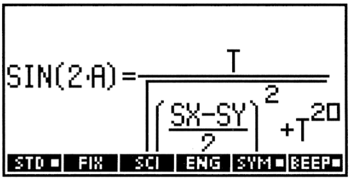
Complete the inner fraction and end the parentheses.

[▲] SX [−] SY [▶]
2 [▶] [▶]



Key in the power, then complete the equation.

[y^x] 2 [▶]
[+] T [y^x] 2



Enter the equation on the stack—it appears in symbolic form.

ENTER

```
1: 'SIN(2*A)=T/√(((SX-
   SY)/2)^2+T^2)'
 STD ■ | FIX | SCI | ENG | SYM ■ | BEEP■
```

Store the equation using the name *MOHR*. (Use α when you type the letters below.)

' MOHR STO

```
2:
1:
 STD ■ | FIX | SCI | ENG | SYM ■ | BEEP■
```

Press VAR to see the *MOHR* label.

To enter an expression or equation in the command line:

1. Press '.
2. Key in the numbers, variables, operators, and parentheses in the expression or equation in left-to-right order. Press ▶ to skip past right parentheses.
3. Press ENTER.

Example: Use the command line to enter the expression $1/(2\pi) \times \sqrt{G/L}$.

Key in the first fraction.

' 1 ÷
↰ ( ) 2 × ↰ π ▶

```
1:
'1/(2*π)'
 MOHR | W | L | N1 |  |
```

Key in the square root term.

× √x ↰ ( ) G ÷ L

```
1:
'1/(2*π)*√(G/L◆'
 MOHR | W | L | N1 |  |
```

Enter the expression on the stack, then store it using the name *FREQ*.

ENTER
' FREQ STO

```
2:
1:
 FREQ | MOHR | W | L | N1 |
```

To edit an expression or equation:

- If it's in level 1, press [↰][EDIT]. After editing, press [ENTER] to save the changes (or press [ATTN] to discard them).
- If it's stored, press ['], press [VAR] `name` for the name, and press [↱][VISIT]. After editing, press [ENTER] to save the changes (or press [ATTN] to discard them).

You can edit expressions and equations the same way you edit other objects. See "Operating the Calculator" on page 1-5 and "Using Memory" on page 1-13.

To do symbolic math:

- To use a command with one argument, key in the algebraic object (the expression or equation), press [ENTER], and press the command key.
- To use a command with two arguments, key in the first algebraic object, press [ENTER], key in the second, press [ENTER], and press the command key.
- To use algebraic objects already on the stack, press the command key.

You make symbolic calculations the same way you make numeric calculations—except you can use algebraic objects instead of just numbers.

Example: Use symbolic math to create the equation $y = 1 - e^{-ax}$.

Starting at the left, enter y and the number 1. (Use [α] when you type the letters in the equation.)

['] Y [ENTER]
1 [ENTER]

Enter the argument $-ax$.

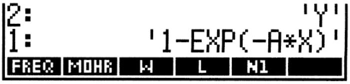
['] [−] A [×] X [ENTER]

Calculate e^{-ax}.

[↰][e^x]

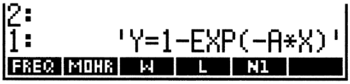
Subtract to calculate $1 - e^{-ax}$.

[−]



Form an equation from the two expressions.

[◄] [=]



To solve an equation for a variable:

1. Enter or recall the expression or equation to level 1 of the stack.
2. Press ['], enter the name of the variable to solve for.
3. Press [◄] [ALGEBRA] ISOL .
4. Optional: To create the variable named on the left-hand side of the equation and store the right-hand expression there, press [◄] [DEF].

ISOL requires that the variable appear in the equation only one time. (If the variable appears more than once, you may be able to simplify the equation—see the instructions following the next example. If the equation is linear or second-order in a variable that appears more than once, you can use QUAD to solve for the variable.)

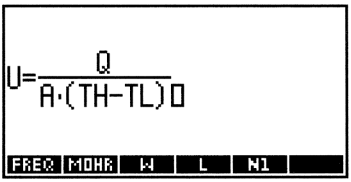
The inverses of many functions have more than one value. If your equation contains such functions, you normally get the *general solution*—it may contain variables such as *n1* or *s1* representing arbitrary integers or signs.

Example: Solve for T_H in the following heat transfer equation, then create variable *TH* containing the resulting expression:

$$U = \frac{q}{A\,(T_H - T_L)}$$

Clear the stack and type the equation. (Use [α] when you type the letters below.)

[►] [CLR]
[◄] [EQUATION]
U [◄] [=] [▲] Q [►]
A [×] [◄] [()] TH [−] TL [►]



Enter the equation and specify the variable name to solve for. (Use (α)
1 when you type the letters below.)

(ENTER)
(') TH

```
1:     'U=Q/(A*(TH-TL))'
'TH♦
FREQ MOHR  W   L   N1
```

Solve for the variable.

(◄)(ALGEBRA) ISOL

```
1:          'TH=Q/U/A+TL'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Create variable *TH* from this equation, then recall it.

(◄)(DEF)
(VAR) TH

```
1:             'Q/U/A+TL'
 TH  FREQ MOHR  W   L   N1
```

To simplify an expression or equation:

1. Press (◄)(ALGEBRA).
2. Simplify the expression or equation:
   - To expand products and powers for further simplification, press EXPA one or more times.
   - To collect and combine like terms, press COLCT.

To evaluate an expression or equation:

1. Enter or recall the expression or equation to level 1 of the stack.
2. Press (EVAL). (To get a strictly numeric result, press (►)(→NUM) instead—or after (EVAL).)

When you evaluate an expression or equation, current values of existing variables are substituted for their names. If a name doesn't exist as a variable, the name isn't replaced. Normally, built-in HP 48 constants, such as π, are evaluated only if you press (►)(→NUM).

Example: Express $\sin(2\pi ft)$ in terms of π and as a numeric value, where f=2100 and t=0.0003. (The calculation assumes the angular units are radians.)

Clear the stack and enter the expression. (Use (α) when you type the letters below.)

(►)(CLR)
(') (SIN) 2 (×) (◄)(π) (×) F (×) T
(ENTER)

```
1:         'SIN(2*π*F*T)'
 TH  FREQ MOHR  W   L   N1
```

Create the variables F and T with the given values. (Use (α) when you type the letters below.)

2100 (ENTER) (') F (STO)
.0003 (ENTER) (') T (STO)

```
1:          'SIN(2*π*F*T)'
 T | F | TH |FREQ|MOHR| W
```

Set Radians mode, then evaluate the expression.

(⬑)(RAD)
(EVAL)

```
1: 'SIN(2*π*2100*.0003
   )'
 T | F | TH |FREQ|MOHR| W
```

Simplify the expression.

(⬑)(ALGEBRA) COLCT

```
1:          'SIN(1.26*π)'
COLCT|EXPA|ISOL|QUAD|SHOW|TAYLR
```

Express the answer as a numeric value. (If you didn't need the answer in terms of π, you could have skipped (EVAL) and COLCT.) Then change back to Degrees mode.

(⮫)(→NUM)
(⬑)(RAD)

```
1:       -.728968627419
COLCT|EXPA|ISOL|QUAD|SHOW|TAYLR
```

Solving Equations for Unknown Values

If you want a numeric solution for an unknown in an equation, you can use the HP Solve application. You can solve for a value of any variable without changing the equation. This means you don't have to solve for the variable symbolically.

If you solve an *equation*, the HP 48 tries to make the difference between the two sides equal to zero. If you solve an *expression*, it tries to make its value zero.

To set up a new equation or expression for solving:

1. Enter the equation or expression on the stack.
2. Press ⬅ SOLVE.
3. Get the equation or expression:
   - To store a copy of the equation or expression for future use, press NEW, type a name for it without pressing α, and press ENTER.
   - To not store a copy, press STEQ.
4. Press SOLVR.

The SOLVR menu that's created has "white" menu labels for variables in the equation or expression. The white labels mean that the SOLVR menu works differently from the VAR menu—as described below.

To store a value in a variable:

1. Enter the value.
2. Press menu key name for the variable name. (Substitute the desired menu key for name.)

To solve for an unknown value:

1. Make sure you store values in all variables. (You don't have to store a value in the one you want to solve for.)
2. Press ⬅ name for the variable name to solve for.

The message Zero or Sign Reversal at the top of the display means a solution was found. A different message means you should evaluate the problem further.

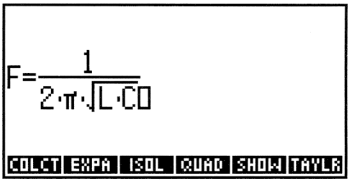
You can solve an equation or expression over and over—for different known values, and for different combinations of known and unknown variables—as shown in the following example.

Example: Find the inductance required for an inductor-capacitor circuit to have a resonant frequency of 18,000 Hz if the capacitance is 33×10^{-6} farads. The equation for this problem is

$$f = \frac{1}{2\pi\sqrt{LC}}$$

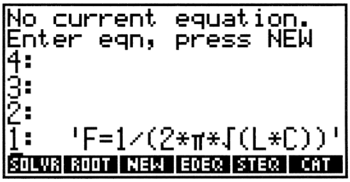
Clear the stack and use the EquationWriter application to key in the equation. (Use [α] when you type the letters below.)

[↱] [CLR]
[↰] [EQUATION]
F [↰] [=] [▲] 1 [▶]
2 [↰] [π] [√x] L [×] C



Put the equation on the stack and get the SOLVE menu.

[ENTER]
[↰] [SOLVE]

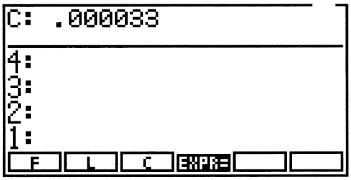


Name the new equation *CKT*, then get the SOLVR menu. (The α annunciator is already turned on, so you shouldn't press [α].)

NEW CKT [ENTER]
SOLVR

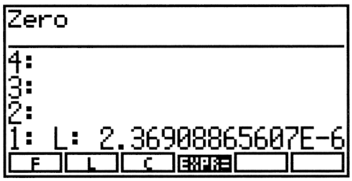
Store the two known values. (Reminder: Keys like F are menu keys, not alpha keys.)

18000 F
33 [EEX] [+/-] 6 C



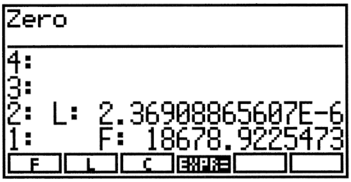
Solve for the inductance.

[↰] L



1 If the closest available value is 2.2×10^{-6}, solve for the actual resonant frequency.

2.2 (EEX) (+/-) 6 [L]
(⬅) [F]



To find a certain solution out of several:

1. Enter a guess for the variable to solve for. The value should be somewhat near the solution you want—at least nearer to the one you want than to other possible solutions.
2. Press [name] for that variable name to store the guess.
3. Press (⬅) [name] for the same variable name.

If the equation or expression has more than one possible solution, the calculator stops when it finds just one. You can find a different solution by storing a guess in the variable you're solving for—it tells the HP Solve application where to start searching.

However, if your problem has several solutions, you should also consider solving it graphically. See the next section, "Getting Answers Graphically."

To look at current variable values:

1. Press (⬅) (REVIEW).
2. Press (ATTN) when you're done.

To get the HP Solve variable menu after an interruption:

- Press (➡) (SOLVE) (*right* shift).

To re-solve an old equation or expression again:

1. Press (⬅) (SOLVE).
2. Press [CAT].
3. Press (▼) and (▲) to move the pointer to the equation or expression you want.
4. Press [SOLVR] to set up that equation or expression for solving.

You can also attach units to the values you're entering and finding. See "Using Numbers with Units" on page 1-44.

Getting Answers Graphically

If you want to see the graphical behavior of an expression or equation and get numeric solutions for one particular variable, you can use the Plot application. You don't have to solve for the variable symbolically.

The information in this section assumes you're making "function" plots—y as a function of x. Other types of plots are possible.

To set up a new expression or equation for plotting:

1. Enter the expression or equation on the stack.
2. Press (←)(PLOT).
3. Get the expression or equation:
   - To store a copy of the expression or equation for future use, press NEW, type a name for it without pressing (α), and press (ENTER).
   - To not store a copy, press STEQ.
4. Verify at the top of the display that the plot type is FUNCTION. If it's not, press PTYPE FUNC.
5. Press PLOTR.

To store a value for a constant:

1. Press (→)(SOLVE) (*right* shift).
2. Enter the value.
3. Press name for the variable name.
4. Press (→)(PLOT) (*right* shift).

Only the independent variable value changes during plotting. All other variables are considered constants.

To plot an expression or equation:

1. Set up the expression or equation for plotting:
2. Press ('), type the name of the independent variable from your expression or equation, and press INDEP.
3. Key in a value for the extreme left end of the horizontal axis, press (SPC) or (ENTER), key in the extreme right value, and press XRNG.
4. Press ERASE.
5. Press AUTO.

The range of the vertical axis is calculated automatically.

If you're plotting an equation with an expression to the left of the equal sign, two curves are plotted—representing the two halves of the equation. If you're plotting an expression—or an equation with just a variable name to the left of the equal sign—only one curve is plotted.

To turn the graphics display on and off:

- To turn off the graphics display, press (ATTN). Then, if you want to return to the plotting setup screen, press (↱)(PLOT) (*right* shift).
- To turn on the graphics display, press (◀) (or (↰)(GRAPH)) while no command line is present.

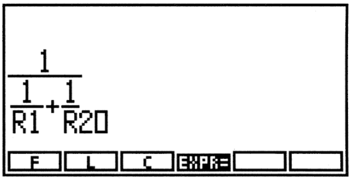
Example: Plot the total resistance of a parallel resistor circuit given by the expression

$$\frac{1}{\frac{1}{R_1}+\frac{1}{R_2}}$$

where R_2 is 1500 ohms, and R_1 varies from 10 to 5000 ohms.

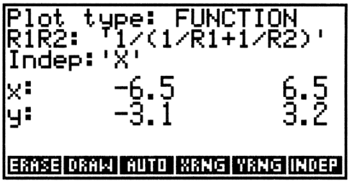
Clear the stack and key in the expression. (Use (α) when you type the letters below.)

(↱)(CLR)
(↰)(EQUATION)
1 (÷) 1 (÷) R1 (▶) (+) 1 (÷) R2



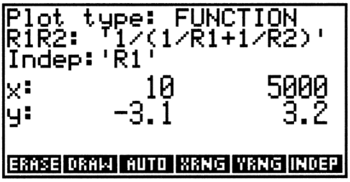
Put the equation on the stack and make it the current equation. (The α annunciator turns on automatically.) Make sure the displayed plot type is FUNCTION. Then set it up as the equation to plot. 1

[ENTER]
[←][PLOT] NEW R1R2 [ENTER]
PTYPE FUNC (if needed)
PLOTR



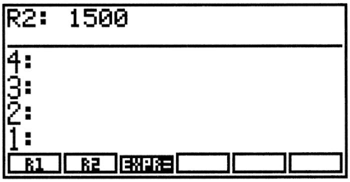
Make *R1* the independent variable, and set its range to 10 to 5000. (Use [α] when you type the letter below.)

['] R1 INDEP
10 [SPC] 5000 XRNG



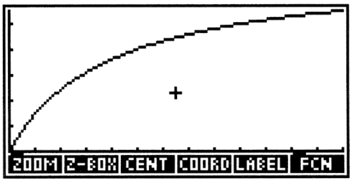
Set *R2* to 1500. (Note the *right* shift.)

[→][SOLVE] 1500 R2



Draw the plot. (Note the *right* shift.)

[→][PLOT] ERASE AUTO



Return to the stack display.

[ATTN]

To estimate coordinates:

1. View the plot in the graphics display.
2. Press [▲], [▼], [◄], and [►] to move the + cursor to the desired point.
3. Press [+] or COORD to see the (x,y) coordinates of the cursor.
4. Press [+] or any menu key to turn off the coordinate display.
5. Optional: Press [ENTER] to put the coordinates on the stack.

To solve for a significant point:

1. View the plot in the graphics display.
2. Press [▲], [▼], [◄], and [►] to move the + cursor near the point of interest.
3. Press FCN .
4. Solve for the coordinates:
   - To find an x value where the curve crosses the x-axis, press ROOT .
   - To find a point where two curves intersect, press ISECT.
   - To find a critical point, such as a maximum or minimum, press EXTR (extremum).
5. Press [ATTN].
6. Press EXIT .

The calculated values from these functions are put on the stack.

Example: The volume of an open tray formed from a 4-by-8 piece of sheet metal is given by $(4-2x)(8-2x)x$, where x is the height of the tray (between 0 and 2). Find the maximum volume and the corresponding x value.

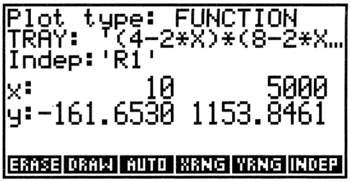
Enter the expression. (Use [α] when you type the letters below.)

['] [←][( )] 4 [−] 2 [×] X [►]
[×] [←][( )] 8 [−] 2 [×] X [►] [×] X
[ENTER]

```
2:
1:  '(4-2*X)*(8-2*X)*X'
ERASE DRAW AUTO XRNG YRNG INDEP
```

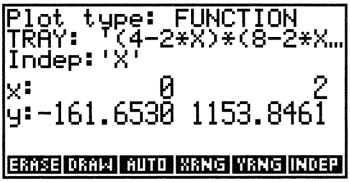
Set up the expression for plotting. (The α annunciator turns on automatically.)

(↰) (PLOT) NEW TRAY (ENTER)
PLOTR



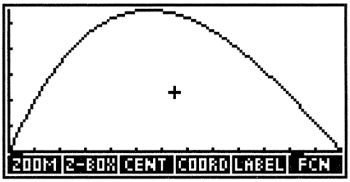
Make *X* the independent variable, and set its range to 0 to 2. (Use (α) when you type the letter below.)

(') X INDEP
0 (SPC) 2 XRNG



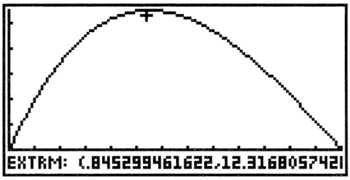
Plot the expression.

ERASE AUTO



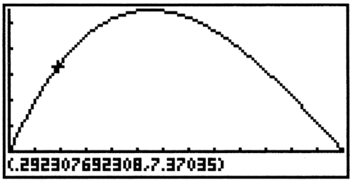
Move the cursor anywhere near the maximum and solve for the coordinates of the maximum. (OFF SCREEN is displayed before the result.)

((◀) and (▲) as needed)
FCN EXTR



Move the cursor to another point on the curve, get the cursor
1 coordinates, and put them on the stack. (Your cursor position may differ.)

([▼] and [◀] as needed)
[+] [ENTER]



Return to the stack display. The (x,y) coordinates for the two points you found are on the stack.

[ATTN]

To get back to the setup menu after an interruption:

- Press [↱] [PLOT] (*right* shift).

To re-plot an old equation or expression:

1. Press [↰] [PLOT].
2. Press CAT .
3. Press [▼] and [▲] to move the pointer to the equation or expression you want.
4. Press PLOTR to set up that equation or expression for plotting.

Making Your Own Functions

If you often make a certain calculation that's not built into the HP 48, you can create a *user-defined* function. Then you can use the new function for numeric *and* symbolic calculations.

To create a user-defined function:

1. Enter an equation that specifies the function name and its arguments on the left side, and the expression that defines the calculation on the right side.
2. Press [↰][DEF].

The syntax for the function definition is '*name*(*arg1* , *arg2*...)=*expression*'. For example, 'AVG3(A,B,C)=(A+B+C)/3' is a valid definition.

To use a user-defined function with the stack:

1. Enter the required argument values on the stack in the same order as they appear in the left side of the function definition. (The last argument should be in stack level 1.)
2. Press [VAR] name for the name of the user-defined function.

To use a user-defined function in an expression:

1. Press ['].
2. Press [VAR] name for the name of the user-defined function.
3. Press [↰][()].
4. Key in the algebraic arguments in their proper order and separated by commas.
5. Press [ENTER] (or press [▶] to continue the expression).

Example: Define a function KE that calculates the kinetic energy of a moving body, given by $\frac{1}{2}mv^2$. Find the kinetic energy for m=14.5 and v=127.9. Also, write an expression for the total energy of two bodies with masses m_1 and m_2 and velocities v_1 and v_2.

Clear the stack and enter the equation that defines the function. (Use [α] when you type the letters below.)

[↱][CLR]
['] KE [↰][()] M [↰][,] V [▶]
[↰][=] .5 [×] M [×] V [y^x] 2 [ENTER]

```
1:   'KE(M,V)=.5*M*V^2'
ERASE DRAW AUTO XRNG YRNG INDEP
```

Create the user-defined function.

1

(⬅)(DEF)

```
1:
ERASE DRAW AUTO XRNG YRNG INDEP
```

Calculate the kinetic energy for the first problem.

(VAR) 14.5 (ENTER) 127.9 KE

```
1:          118598.4725
 KE    X   TRAY  R1   R2  R1R2
```

Write the expression for the total kinetic energy. (Use (α) when you type the letters below.)

(') KE (⬅)(( )) M1 (⬅)(,) V1 (▶)
(+) KE (⬅)(( )) M2 (⬅)(,) V2
(ENTER)

```
2:          118598.4725
1: 'KE(M1,V1)+KE(M2,V2
   )'
 KE    X   TRAY  R1   R2  R1R2
```

If you stored values in variables *M1*, *M2*, *V1*, and *V2*, you could evaluate this expression.

Programming

For repetitive problems that aren't suited to symbolic expressions or other techniques, you can create programs. You can use programs to perform any sequence of operations you want. The keys you press for a keyboard calculation represent a series of commands—you can include those commands in a program to do the same calculation.

A program is simply a sequence of commands, numbers, and other objects that are processed in order. A program is an object—so it occupies one level of the stack, and you can store it in a variable. However, a complete "program" often consists of several programs that work together—much like subroutines.

You create programs as described below. You store and recall them the same way you store and recall other objects—see "Using Memory" on page 1-13.

To enter a program:

1. Press [⬅][« »].
2. Enter the commands, numbers, and other objects in the order you want them processed.
   - For a command, press the command key or type its name.
   - For a number, type the number. Press [SPC] to separate two consecutive numbers.
   - For a variable, press [VAR] [name] or type its name.
   - For a symbolic expression, press ['], enter the expression, and press [▶].
   - For any other object, type its delimiters, enter its contents, and press [▶].
3. Press [ENTER].

Lines you enter in a program can be as long or short as you want. You can start a new line any time by pressing [➡][↵] (newline). Your line breaks are discarded when you press [ENTER].

For calculations, you can use "stack" calculations (entering data and executing commands) or "symbolic" calculations (entering expressions and evaluating them). They can look quite different in a program, but can give the same results.

In a program, a variable name behaves the same as when you press the corresponding key in the VAR menu.

Example: Create a program that squares two numbers from levels 1 and 2 of the stack, then finds the absolute value of the difference: $|x_1{}^2 - x_2{}^2|$.

In order, the program squares the number in level 1, swaps the values in levels 1 and 2, squares the new number in level 1, subtracts the values in levels 1 and 2, and finds the absolute value of the result.

Clear the stack and start entering the program.

[➡][CLR]
[⬅][« »] [⬅][x^2] [⬅][SWAP] [⬅][x^2]

Complete the program.

[−] [MTH] [PARTS] [ABS]

Put the program on the stack.

[ENTER]

```
2:
1: « SQ SWAP SQ - ABS
   »
 ABS | SIGN | CONJ | ARG | RE | IM
```

Store the program in variable *DIFF*. (Use [α] when you type the letters below.)

['] DIFF [STO]

```
1:
 ABS | SIGN | CONJ | ARG | RE | IM
```

To enter a simple program with local variables:

1. Press [↰] [« »].
2. Press [↱] [→].
3. Enter names for one or more local variables. Press [SPC] to separate names.
4. Press ['] and enter a symbolic expression that uses the local variable names.
5. Press [ENTER].

Local variables are temporary variables. The → command takes numbers or other objects from the stack and stores them in the temporary variables—then those values are used to evaluate the symbolic expression. By using local variables and a symbolic expression, you have a program that's easy to create and understand.

Example: Rewrite the previous *DIFF* program so that it uses local variables to calculate $|{x_1}^2 - {x_2}^2|$.

Enter the program. (Use [α] when you type the letters in this example.)

[↰] [« »] [↱] [→] X2 [SPC] X1 [SPC] ['] [MTH] PARTS ABS X1 [y^x] 2 [−] X2 [y^x] 2 [ENTER]

```
2:
1: « → X2 X1 'ABS(X1^2
   -X2^2)' »
 ABS | SIGN | CONJ | ARG | RE | IM
```

Store the program in variable *DIFF*.

[VAR] [↰] DIFF

```
1:
 DIFF | KE | X | TRAY | R1 | R2
```

To execute a program:

- Press (VAR), then press name for the variable name where the program is stored.
  or
- Key in the variable name where the program is stored (without (') tick marks) and press (ENTER).
  or
- Put the program in level 1 of the stack and press (EVAL). (The program is removed from the stack, then it starts executing.)

If you store a program in a variable, you can execute it by name in any other program—just include the variable name.

To interrupt an executing program:

- Press (ATTN).

Example: Use the previous *DIFF* program to find $|6^2 - 9^2|$.

9 (ENTER) 6 (VAR) DIFF

To edit a program:

- If it's in level 1, press (◄)(EDIT). After editing, press (ENTER) to save the changes (or press (ATTN) to discard them).
- If it's stored, press ('), press (VAR) name for the name, and press (►)(VISIT). After editing, press (ENTER) to save the changes (or press (ATTN) to discard them).

You edit programs the same way you edit other objects. See "Operating the Calculator" on page 1-5 and "Using Memory" on page 1-13.

1

Using Numbers with Units

Many physical problems involve values with associated measurement units, such as 17.5 *meters* and 324 *calories per second*. The HP 48 lets you attach units of measure to numeric values. Such combinations are called *unit objects*. The HP 48 provides more than 100 built-in units—and you can combine them at will into compound units.

To include units with a number:

1. Enter the number. You don't have to press [ENTER].
2. Press [◀] [UNITS].
3. Press [NXT] as required, then press the menu key for the appropriate category of units.
4. Press [NXT] as required, then press `unit` for the units you want. Press [▶] `unit` instead if you want the *inverse* of the units. (Substitute the desired unit menu key for `unit`.)
5. For compound units, repeat steps 2 through 4 for each individual unit in the compound unit.

Example: For the element silicon, its atomic radius is 1.46 angstroms and its density is 2.33 grams per cubic centimeter. Enter these values.

Clear the stack and enter the atomic radius.

[▶] [CLR]
1.46 [◀] [UNITS] `LENG`
[NXT] [NXT] [NXT] `Å`

Start entering the density.

2.33 [◀] [UNITS] `MASS` `G`

Complete the compound units for density.

[◀] [UNITS] `VOL` [▶] `CM^3`

To calculate with units:

1. Enter values with units.
2. Execute commands.

Units are automatically converted and combined during the calculation—you don't have to do any additional work.

However, you must use *consistent units* for certain operations, such as addition. Consistent units are units that have the same physical dimensions, such as length or density. For such operations, the answer is automatically converted to the units from the value in level 1.

Example: Find the final velocity of an object in free fall after 8 seconds if it starts with an upward velocity of 5 centimeters per second. The final velocity is calculated as $v_0 - gt$, where g is 9.8 m/s^2.

Clear the stack and enter values with units for v_0, g, and t.

```
↱ CLR
5 ↰ UNITS  SPEED  CM/S
9.8 ↰ UNITS  LENG  M
↰ UNITS  TIME
↱ S  ↱ S
8  S
```

```
3:              5_cm/s
2:            9.8_m/s^2
1:                 8_s
 YR | D | H | MIN | S | HZ
```

Multiply g and t.

```
×
```

```
2:              5_cm/s
1:           78.4_m/s
 YR | D | H | MIN | S | HZ
```

To get the answer with the same units as v_0, you want to have v_0 in level 1 when you combine terms. So make gt negative, exchange the values in levels 1 and 2, then add the values to get the final velocity.

```
+/-  ▶  +
```

```
1:          -7835_cm/s
 YR | D | H | MIN | S | HZ
```

To convert units to a built-in unit:

1. Enter the value with its original units.
2. Press ↰ UNITS.
3. Press the menu key for the appropriate category of units.
4. Press ↰ unit for the units you want to convert to.

Example: Convert 14 cm/s to mi/hr.

Clear the stack, and enter and convert the value.

[↱] [CLR]
14 [↰] [UNITS] SPEED CM/S
[↰] MPH

```
1:    .313171080888_mph
 M/S | CM/S | FT/S | KPH | MPH | KNOT
```

To convert to any units:

1. Enter the value with its original units.
2. Enter any number (such as 1) and attach the units you want to convert to.
3. Press [↱] [UNITS] (*right* shift).
4. Press CONV.

The numeric value of the second argument is ignored during the conversion.

Example: Convert 9.8 m/s^2 to ft/s^2.

Clear the stack and enter the value.

[↱] [CLR]
9.8 [↰] [UNITS] SPEED M/S
[↰] [UNITS] TIME [↱] S

```
1:              9.8_m/s^2
 YR | D | H | MIN | S | HZ
```

Enter a number with the desired units. (Because the denominator units are in the current menu, enter them first.)

1 [↱] S [↱] S
[↰] [UNITS] LENG FT

```
2:              9.8_m/s^2
1:               1_ft/s^2
 M | CM | MM | YD | FT | IN
```

Convert the units. (Note the *right* shift.)

[↱] [UNITS] CONV

```
1: 32.1522309711_ft/s^
   2
 CONV | UBASE | UVAL | UFACT | →UNIT |
```

To delete units from the number in level 1:

1. Press [↱] [UNITS] (*right* shift).
2. Press UVAL (unit value).

To use units with the HP Solve application:

1. Store values with appropriate units in *all* variables—including a guess with units for the unknown variable.
2. Solve the equation.

The HP 48 automatically converts units during the process, and it converts the solution to the units you specified. If any units are incompatible during the calculation, an error occurs.

The HP Solve menu keys automatically reuse current units for variables. To change a variable value without changing the units, store just the numeric value. To change the value *and* units, store a value with units. To delete units from a variable, enter the numeric value, then press (VAR) (←) name.

Keeping Track of Time

You can use the built-in clock to get the time and date, to set alarms, and to do other time-related operations.

To change the time or date format:

1. Press (←)(TIME).
2. Press SET.
3. Set the format:
   - To change the date format between *month/day/year* and *day.month.year*, press M/D.
   - To change the time format between 12-hour (AM and PM) and 24-hour (no AM or PM), press 12/24.

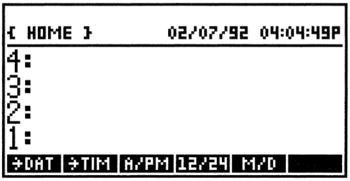
To set the time and date:

1. Press ⬑ TIME.
2. Press SET.
3. Enter the date as *MM.DDYYYY* or *DD.MMYYYY* (depending on the current date format), where *MM* is the month (01 to 12), *DD* is the day (01 to 31), and *YYYY* is the year (such as 1991).
4. Press →DAT.
5. Enter the time as *HH.MMSSsss* (12-hour or 24-hour), where *HH* is the hour (0 to 24), *MM* is the minutes (00 to 59), and *SSsss* is the integer and decimal seconds (0 to 59999 ...).
6. Press →TIM.
7. To change the time between AM and PM, press A/PM.

Example: Set the time and date to 3:25 PM on February 7, 1992—or to the current time and date, if you want. (This example assumes month/day/year and 12-hour formats.)

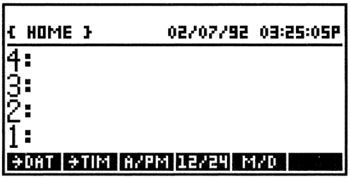
Clear the stack and set the date—use today's actual date, if you want.

⬏ CLR
⬑ TIME SET 2.071992 →DAT



Set the time—use the current time, if you want.

3.25 →TIM A/PM



To see the time and date:

- Press ⬑ TIME. (The time and date turn off when you leave the TIME menu.)

To turn the permanent time-and-date display on and off:

1. Press ⬅ [MODES].
2. Press [NXT].
3. Press CLK .

To set an alarm:

1. Press ⬅ [TIME].
2. Press ALRM .
3. To set an alarm date different from today, enter the date as *MM.DDYYYY* or *DD.MMYYYY* (depending on the current date format), then press >DATE.
4. Enter the alarm time as *HH.MMSSsss* (12-hour or 24-hour), then press >TIME.
5. To change the alarm time between AM and PM, press A/PM .
6. Press ➡ [" "], enter a short message as a text string, then press EXEC .
7. Press SET .

When the alarm time arrives, the HP 48 beeps for several seconds, the message is displayed, and the ((•)) annunciator turns on.

To respond to an alarm:

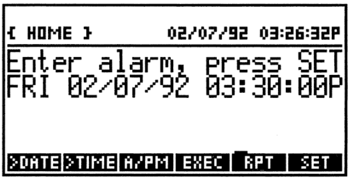
- During the beeps, press any key, such as [ATTN].

  or
- After the beeps stop, press ⬅ [TIME] to see the alarm message, then press ACK [ATTN].

Example: Set an alarm for a few minutes from now.

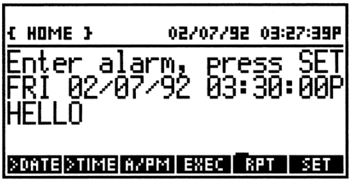
Set the alarm time—you don't have to change the date. (Use an alarm time that's a few minutes from the time in your display.)

⬅ [TIME] ALRM
3.30 >TIME A/PM



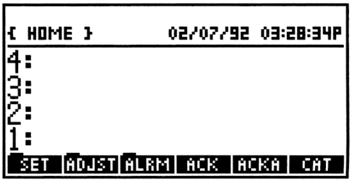
Enter a message. (Use (α) when you type the letters below.)

(↱)(" ") HELLO EXEC



Set the alarm and return to the stack display.

SET (ATTN)



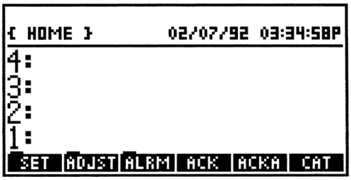
Wait until the alarm occurs and the beeps stop. The alarm time and message are displayed only during the beeps. The ((•)) annunciator stays on (though it's not shown below).

View the alarm message.

(↰)(TIME)

Acknowledge the alarm.

ACK (ATTN)



Press (MTH) to return to the MTH menu.

2

The Keyboard and Display

This chapter describes the keyboard and display in detail. It shows how to enter information and how to understand displayed information.

Organization of the Display

For most operations, the display is divided into three sections. This configuration is called the *stack display*. Each section is described in the following topics.

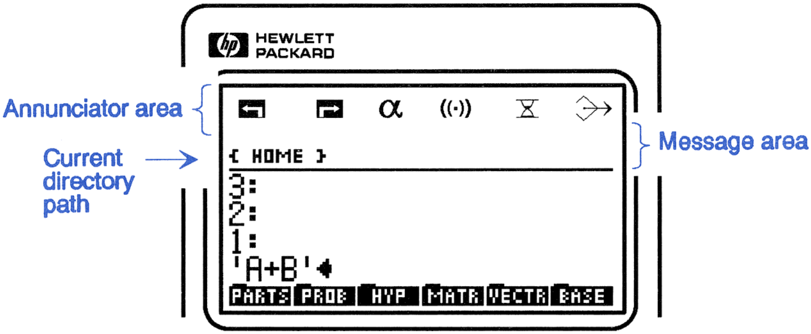
The Status Area, Annunciators, and Messages

The status area displays:

- **Annunciators.** They indicate the current status of the calculator.
- **The current directory path.** When you turn the calculator on for the first time, the current directory path is { HOME }. Directories divide memory into parts, much as files do in a file cabinet—they're covered in chapter 7.

- **Messages.** They inform you when an error has occurred, or provide other information to help you use the calculator more effectively.

In the table that follows, the first six annunciators appear at the very top of the display. The other annunciators and the directory path share their "territory" with messages—a message replaces the annunciators and directory path. When you clear the message, the directory path and any active annunciators reappear.



Annunciators

Symbol	Meaning
	Left-shift is active (you pressed ⬅).
	Right-shift is active (you pressed ➡).
α	The alpha keyboard is active (you can type letters and other characters).
((•))	Alert. An appointment has come due or a low battery condition has been detected. See the message in the status area for information. (If no message displayed, turn the calculator off and on. A message describing the cause of the alert should appear.)
⧗	Busy—not ready to process new input. However, the calculator can remember up to 15 keystrokes while busy and then process them when free.
	Transmitting data to an external device.

Annunciators (continued)

Symbol	Meaning
RAD	Radians angle mode is active.
GRAD	Grads angle mode is active.
R∡Z	Polar/Cylindrical coordinates mode is active.
R∡∡	Polar/Spherical coordinates mode is active.
HALT	Program execution has been halted.
1 2 3 4 5	The indicated user flags are set.
1USR	The user keyboard is active for one operation.
USER	The user keyboard is active until you press (↰)(USR).
ALG	Algebraic-entry mode is active.
PRG	Program-entry mode is active.

The Stack

The stack is a series of storage locations for numbers and other objects. The locations are called levels 1, 2, 3, etc. The number of levels changes according to how many objects are stored on the stack—from none to hundreds or more.

As you enter new numbers or other objects on the stack, the stack grows to accommodate them—new data moves into level 1, and older data is "bumped" to higher levels. As you use data from the stack, the number of levels decreases as the data moves down to lower levels.

The stack display shows level 1 and up to three additional levels. Any additional levels are maintained in memory, but you normally can't see them.

For more information about the stack and command line, see "Using the Stack for Calculations" on page 3-2.

2

The Command Line

The command line appears whenever you start keying in or editing text. The stack lines move up to make room. If you type more than 21 characters, information scrolls off the left side of the display, and an ellipsis (...) appears to tell you there is more information "in that direction."

After you finish using the command line, the stack display moves down into the command line area.

For more information about the stack and command line, see "Using the Command Line" on page 3-15.

Menu Labels

Menu labels across the bottom of the display show the operations associated with the six white menu keys across the top of the keyboard. See "Working with Menus" on page 2-11 for information about using menus.

Organization of the Keyboard

The HP 48 keyboard has six levels or "layers," each containing a different set of keys:

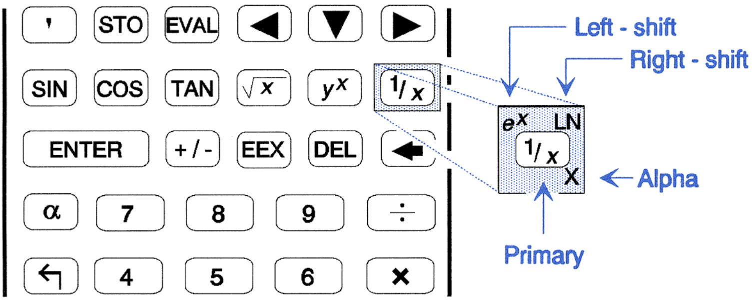
- **Primary keyboard**, represented by the labels on the key faces. For example, [+], [7], [ENTER], [TAN], and [▲] are all keys on the primary keyboard.
- **Left-shift keyboard**, activated by pressing the orange [↰] key on the primary keyboard. A left-shift key is labeled in orange and located above and to the left of its associated primary key. To execute ASIN, for example, you press the [↰] key followed by the associated [SIN] key.
- **Right-shift keyboard**, activated by pressing the blue [↱] key on the primary keyboard. A right-shift key is labeled in blue and located above and to the right of its associated primary key. To execute →NUM, for example, you press the [↱] key followed by the associated [EVAL] key.

■ **Alpha keyboard**, activated by pressing the (α) key on the primary keyboard. An alpha key is labeled in white and located to the right of its associated primary key. Alpha keys are all capital letters. To generate "N", for example, you press (α) followed by the associated (STO) key. When the alpha keyboard is active, the number pad generates its primary numeric characters.

■ **Alpha left-shift keyboard**, activated by pressing (α) and then orange (⬅) on the primary keyboard. Alpha left-shift characters are primarily lowercase letters, along with some special characters. To type "n", for example, you press (α), then (⬅), and then (STO). (Alpha left-shift characters are not shown on the keyboard.)

■ **Alpha right-shift keyboard**, activated by pressing (α) and then blue (➡) on the primary keyboard. Alpha right-shift characters are Greek letters and other special characters. To generate λ, for example, you press (α), then (➡), and then (NXT). (Alpha right-shift characters are not shown on the keyboard.)

The unshifted and shifted Alpha keyboards are shown on page 2-8.



When you press (⬅) (left-shift) or (➡) (right-shift) to access the shifted operations printed above the primary keys, the ⬅ or ➡ annunciator turns on to indicate that left-shift or right-shift is active.

To cancel a shift key:

■ To change to the *other* shift key, press the other shift key.
■ To just clear the shift key, press the shift key again.

Using the Keyboard and Display

Getting Attention!

When the HP 48 is on, [ON] becomes the [ATTN] (attention!) key. Generally, [ATTN] halts the current activity—so you can immediately start your next task or recover from an unexpected situation.

To stop whatever's happening:

- To delete the command line, press [ATTN].
- To cancel a special environment and restore the stack display, press [ATTN].
- To cancel a running program, press [ATTN].

Keying In Numbers

The basic type of data you'll use is numbers. Although the HP 48 accepts different types of numbers (real numbers, complex numbers, vectors, etc.), you key in all numeric values the same way. When you key in numbers in the stack display, they're displayed in the command line.

To key in a simple number:

1. Press the appropriate number and [.] keys.
2. If the number is negative, press [+/-].

To delete the command line:

- Press [ATTN].

Example: Enter the number −123.4 in the command line.

Key in the digits.

123 [.] 4

Make the number negative.

[+/-]

Press (ATTN) (the (ON) key) to delete the command line.

To correct a typing mistake:

- Press (◆) (the backspace key) to erase the mistake, then retype it correctly.

To key in a number as a mantissa and an exponent:

1. Key in the mantissa. If it's negative, press (+/-) to change its sign.
2. Press (EEX). (It types an E for "exponent.")
3. Key in the exponent—the power of 10. If it's negative, press (+/-).

Example: Enter 1.2×10^{-3}.

Enter the mantissa (1.2).

1.2

Enter the exponent (−3).

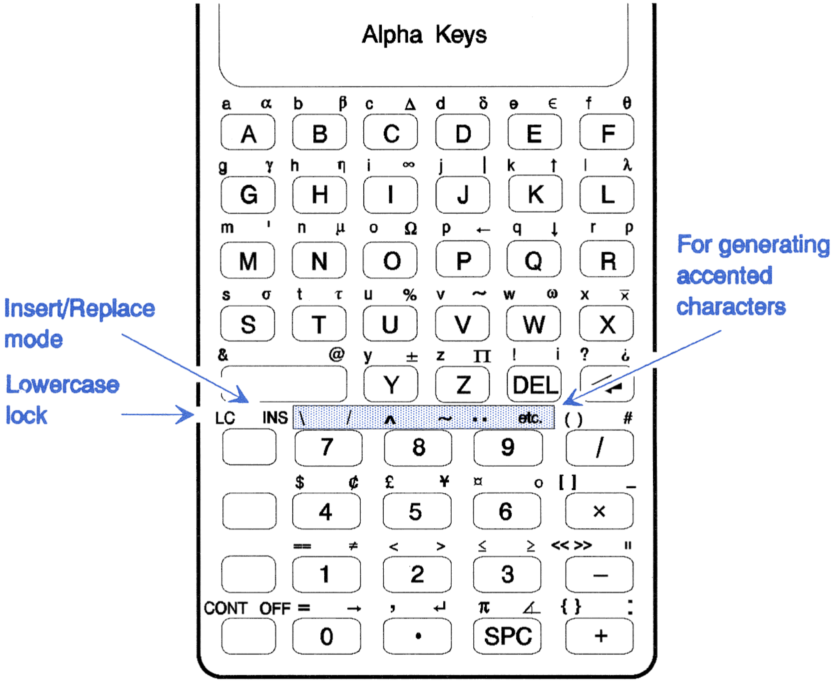
(EEX) 3 (+/-)

Press (ATTN) to delete the command line.

Keying In Characters (the Alpha Keyboard)

Whenever you key in letters and other characters, you use the alpha keyboard. The α annunciator turns on whenever the alpha keyboard is active—whenever Alpha-entry mode is active. Characters you type in the stack display are displayed in the command line.

The primary (unshifted) alpha assignments are printed on the keyboard to the lower right of each key. In addition, many keys have left- and right-shifted alpha assignments. (All uppercase letters are unshifted, while their lowercase counterparts are left-shifted.) To keep the HP 48 keyboard from appearing too cluttered, most of the alpha left- and right-shift keys are not shown on it. For your reference, the next illustration shows the unshifted and shifted alpha keys.



To key in one character:

- Press (α) and key in the character.

or

- Hold down (α), key in the character, then release (α).

To key in several characters:

- Press (α)(α), key in the characters, press (α).

or

- Hold down (α), key in the characters, then release (α).

Pressing (α) *one time* activates Alpha-entry mode for one character. For example, pressing (α) then (SIN) types S.

Pressing (α) *twice in a row* locks Alpha-entry mode. Alpha-entry mode remains active until you press (α) again or press (ENTER).

You can press and hold down (α) while you type several characters in a row—the α annunciator turns off when you release (α).

To lock or unlock the lowercase keyboard:

- If **α** is locked on, press [↰][α].
- If **α** is off, press [α][↰][α].

Only letter keys are affected while lowercase Alpha-entry mode is locked. To get uppercase letters, you must use [↰]. Lowercase mode automatically unlocks when you press [ENTER], [ATTN], or execute a command.

Example: Type the phrase "HP 48 power!" in the command line. (This example shows one sequence of [α] keystrokes, though you can use a different sequence as mentioned above.)

Type the quote marks and *HP 48*. The **α** can be on or off while typing the number characters.

[↱][" "] [α] H [α] P [SPC] 48

Type the space, lock Alpha-entry mode and lowercase, and type *power*.

[SPC] [α][α] [↰][α] power

Type the exclamation mark.

[↰][DEL]

Press [ATTN] to delete the command line and cancel Alpha-entry mode.

To key in an accented character:

1. Type the base character (without an accent).
2. Press the accent key (immediately after the base character).

To generate an accented character during editing, position the cursor to the right of the letter and then type the accent. The letter to the immediate left is changed.

The HP 48 provides five accent marks (`, ´, ˜, ˆ, and ¨) that you can use with appropriate letters. In addition, the alpha [→][etc] key works in conjunction with certain other letters. These six keys are the alpha left- and right-shift keys associated with the primary keys [7], [8], and [9]. (See the alpha-keyboard diagram on page 2-8.)

You can use the accent marks and the [→][etc] key to generate other special characters, as shown in the following table.

Use [→][etc]		Use Any Accent Mark	
To Change:	**To:**	**To Change:**	**To:**
A	Å	C	Ç
a	å	c	ç
E	Æ	D	Đ
e	æ	d	đ
O	Ø	P	Þ
o	ø	p	þ

Example: Key in the characters ý and Ø.

Key in the lowercase letter y. (Use [α] [←] [+/−].)

y

Key in the ´ accent character. (Use [α] [→] [7].)

´

Key in the uppercase letter O.

O

Change the O to Ø. (Use [α] [→] [9].)

[α] [→][etc]

Press [ATTN] to delete the command line and cancel Alpha-entry mode.

Note

Keystroke examples in the rest of this manual show the alpha characters without the [α] key. For example, the keystrokes for entering "HELLO" onto the stack are shown as [↱] [" "] HELLO [ENTER]. Even though it's not shown in the keystrokes, you still must activate Alpha-entry mode in one of the ways described above before entering the alpha characters.

Keying In Objects with Delimiters

Real numbers represent one type of object. Most other types of objects require special *delimiters* to indicate the type of object. For example, a text string requires " " delimiters.

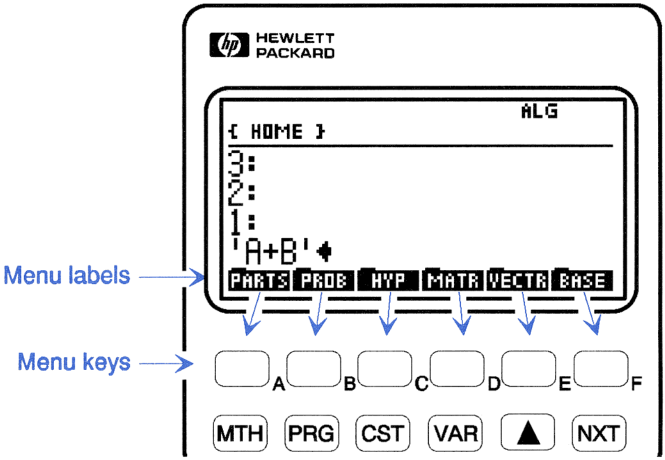
To key in an object with delimiters:

- For opening and closing delimiters, press the delimiter key, then key in the data. (The delimiter key types *both* delimiters.)
- For a single delimiter, press the delimiter key where required in the data.

Objects and delimiters are described in chapter 4.

Working with Menus

A menu is a set of operations defined for the six blank *menu keys* at the top of the keyboard. The current operations are described by the six *menu labels* at the bottom of the display.



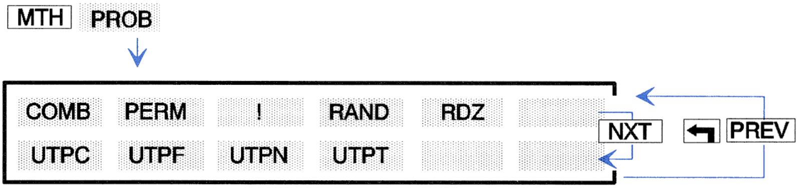
Some menus have multiple sets of labels, called *pages*. If a menu label has a bar over the left corner, it selects another menu—a *submenu*.

To display a menu:

1. Press the key or keys corresponding to the menu you want.
2. If necessary, change to the menu page you want:
   - To move to next page, press [NXT].
   - To move to the previous page, press [↰][PREV].
   - To move to the first page, press [↱][PREV] (*right* shift).

Many of the keys on the HP 48 keyboard display menus: [MTH], [PRG], [CST], [VAR], and all of the shifted keys with orange labels on darker backgrounds (such as [↰][MODES]).

For menus with more than six entries, you can cycle through its pages, eventually returning to the first page. In the following illustration of the MTH PROB menu, notice how [NXT] and [↰][PREV] work.



With a few exceptions, when you want to go to another menu, simply press the keys for that menu—you don't "get out" or "back out" of one menu to go to another—you just go to the new one. (The special menus that act a little differently are explained in later chapters.)

Example: Get the MTH menu and notice that each menu key has a bar over its left corner and, therefore, calls another menu.

MTH	PARTS PROB HYP MATR VECTR BASE

Get the PROB submenu and display the second page.

PROB NXT	UTPC UTPF UTPN UTPT

Return to the MTH menu.

MTH	PARTS PROB HYP MATR VECTR BASE

To switch to the previous menu:

- Press ⮫ LAST MENU.

There may be times when you are working primarily with a particular menu, but need to use commands in another menu. For example, you may need to leave briefly the third page of the STAT menu to use a command in the second page of the MTH PROB menu.

When you switch from one menu to another, the HP 48 stores the identity and page number of the last menu you were in. Pressing ⮫ LAST MENU (found over the 3 key) returns you to that menu. Menus of menus (such as the MTH menu) aren't stored as the last menu.

To perform a menu operation:

- Press the menu key below the label for the operation.

Example: Display the MTH PROB (math probability) menu and calculate 7! (7 factorial).

Key in the 7 and execute the factorial function.

7 [MTH] PROB !

Setting the Display Mode

The *display mode* controls the format the HP 48 uses to display numbers. (Regardless of the current display mode, a number is always stored as a signed, 12-digit mantissa with a signed, 3-digit exponent.) The keys for setting the display mode are located in the MODES menu ([←][MODES]). A ■ in the menu label indicates the mode is active—for instance, STD■ means Standard mode is active.

To set the display mode:

- For standard format, press [←][MODES] STD.
- For *n* decimal places, enter the number *n*, then press [←][MODES] FIX.
- For scientific format, enter the number of decimal places, then press [←][MODES] SCI.
- For engineering format, enter the number of digits after the first one, then press [←][MODES] ENG.
- To change the fraction mark to period or to comma, press [←][MODES] [←][PREV] FM,.

Display Modes

Key	Programmable Command	Description
(MODES) (pages 1 and 4):		
STD	STD	**Standard mode.** Displays numbers using full precision. All significant digits to the right of the decimal point are shown, up to 12 digits.
FIX	FIX	**Fix mode.** Displays numbers rounded to a specified number of decimal places. Real numbers on the stack are displayed with digit separators—commas (for period fraction mark) or periods (for comma fraction mark). Uses a number from the stack for the number of decimal places.
SCI	SCI	**Scientific mode.** Displays a number as a mantissa (with one digit to the left of the decimal point) and an exponent. Uses a number from the stack for the number of decimal places in the mantissa.
ENG	ENG	**Engineering mode.** Displays a number as a mantissa followed by an exponent that is a multiple of 3. Uses a number from the stack for the number of mantissa digits to be displayed after the first significant digit.
FM,		**Fraction mark.** Switches fraction mark (the character that separates the integer and fractional part of the number) between period and comma. ▪ in the label indicates a comma fraction mark.

Example: Key in 12345.6789 and show it in various display modes, starting with two decimal places.

12345.6789 (ENTER)
2 (MODES) FIX

Switch to scientific notation with a 5-decimal-place mantissa.

5 SCI

Switch to engineering notation with a 4-digit mantissa (3 digits after the first digit).

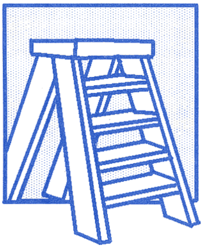
3 ENG

Return to Standard display mode.

STD

3

The Stack and Command Line



The stack is a series of storage locations for numbers and other objects. The locations are called levels 1, 2, 3, etc. The number of levels changes according to how many objects are stored on the stack—the number is limited only by the amount of memory available. If the stack is empty, *no* data is available there—not even zeros.

In general, you enter numbers and other objects onto the stack, and then execute commands that operate on the data. As you enter new objects, the stack grows to accommodate them—new objects move into level 1, and older objects are "bumped" to higher levels. As you use objects, they're removed from the stack, the number of levels decreases, and the remaining objects move down to lower levels. You usually work with only the first few levels of the stack—higher levels hold objects that you can use as needed.

The stack display shows level 1 and up to three additional levels. Any additional levels are maintained in memory, but you normally can't see them.

The command line is closely tied to the stack. You use it to key in (or edit) text and then to process it, transferring the results to the stack.

This chapter covers:

- Using the stack for calculations.
- Viewing and editing the contents of the stack.
- Using the command line.

Using the Stack for Calculations

You do ordinary calculations by entering objects onto the stack and then executing the appropriate functions and commands. The fundamental concepts of stack operations are:

- A command that requires *arguments* (objects the command acts upon) takes its arguments from the stack. (Therefore, the arguments must be present *before* you execute the command.)
- The arguments for a command are removed from the stack when the command is executed.
- Results are returned to the stack so that you can see them and use them in other operations.

Making Calculations

When you execute a command, any arguments in the command line are put onto the stack *before* the command is executed. So you don't have to press [ENTER] to put the arguments on the stack—you can leave one or more arguments in the command line when you execute the command. (You should still think of the arguments as being on the stack, though.)

To use a one-argument command:

- Enter the argument into level 1 (or into the command line).
- Execute the command.

Example: Use the one-argument commands LN ([↱][LN]) and INV ([1/x]) to calculate 1/ln 3.7.

3.7 [↱][LN]
[1/x]

```
1:            .764331510286
STO  FIX  SCI  ENG  SYM  BEEP
```

To use a two-argument command:

- Enter the first argument and then the second argument. The first argument should be in level 2, and the second in level 1 (or in the command line).
- Execute the command.

A two-argument command acts on the arguments (objects) in levels 1 and 2, and returns the result to level 1. The rest of the stack *drops* one level—for example, the previous contents of level 3 move to level 2. The arithmetic functions (+, −, ×, /, and ^) and percent calculations (%, %CH, and %T) are examples of two-argument commands.

Example: Calculate 85 − 31.

85 [ENTER] 31 [−]

Example: Calculate $\sqrt{45} \times 12$.

45 [$\sqrt{x}$] 12 [×]

Example: Calculate $4.7^{2.1}$.

4.7 [ENTER] 2.1 [y^x]

To enter more than one argument in the command line:

- Press [SPC] to separate arguments.

Example: Calculate $\sqrt[4]{2401}$.

2401 [SPC] 4 [→] [$\sqrt[x]{y}$]

To use previous results (chain calculation):

- If necessary, move the previous results to the proper stack level for the command.
- Execute the command.

Chain calculations involve more than one operation. The stack is especially useful for chain calculations because it retains intermediate results.

Example: Calculate $(12 + 3) \times (7 + 9)$.

12 [ENTER] 3 [+]
7 [ENTER] 9 [+]

3

Notice that the two intermediate results remain on the stack. Now, multiply them.

[×]

```
1:                 240
```

Example: Calculate $23^2 - (13 \times 9) + \frac{5}{7}$. First, calculate 23^2 and the product 13×9.

23 [←][x^2]
13 [ENTER] 9 [×]

```
2:                 529
1:                 117
```

Subtract the two intermediate results and calculate $\frac{5}{7}$.

[−]
5 [ENTER] 7 [÷]

```
2:                 412
1:       .714285714286
```

Add the two results.

[+]

```
1:       412.714285714
```

Manipulating the Stack

To swap the objects in levels 1 and 2:

- Press [←][SWAP] (or [▶] when no command line is present).

The SWAP command is useful with commands where the order is important, such as −, /, and ^ ([−], [÷], and [y^x]).

Example: Use [←][SWAP] to help calculate $\frac{9}{\sqrt{13+8}}$. First, calculate $\sqrt{13+8}$.

13 [ENTER] 8 [+] [$\sqrt{x}$]

```
1:       4.58257569496
```

Enter 9 and swap levels 1 and 2.

9 [←][SWAP]

```
2:                   9
1:       4.58257569496
```

Divide the two values.

[÷]

```
1:       1.96396101212
```

To duplicate the object in level 1:

- Press (PRG) STK (NXT) DUP (or press (ENTER) if no command line is present).

The DUP command duplicates the contents of level 1 and pushes the other stack contents up one level.

Example: Calculate $\frac{1}{47.5} + \left(\frac{1}{47.5}\right)^4$. First, calculate the inverse of 47.5 and duplicate the value.

47.5 (1/x) (ENTER)

```
2:     2.10526315789E-2
1:     2.10526315789E-2
STD  FIX  SCI  ENG  SYM  BEEP
```

Raise the value to the 4th power.

4 (y^x)

```
2:     2.10526315789E-2
1:     1.96438026103E-7
STD  FIX  SCI  ENG  SYM  BEEP
```

Add the result to the original value.

(+)

```
1:     2.10528280169E-2
STD  FIX  SCI  ENG  SYM  BEEP
```

To delete the object in level 1:

- Press (⬅)(DROP) (or (◀) when no command line is present).

When you execute the DROP command, the remaining objects on the stack drop down one level.

To clear the entire stack:

- Press (➡)(CLR) (the CLEAR command).

Recalling the Last Arguments

To recall the arguments of the last command:

- Press (➡)(LAST ARG) (the LASTARG command).

The LASTARG command places the arguments of the most recently executed command on the stack so that you can use them again.

Example: Use [↱][LAST ARG] to help calculate ln 2.3031 + 2.3031. First, calculate ln 2.3031, then retrieve the argument of LN. ([LAST ARG] is the blue, right-shifted label above the [2] key.)

2.3031 [↱][LN]
[↱][LAST ARG]

Add the two numbers.

[+]

LASTARG is particularly useful for more complicated arguments, such as matrices.

Restoring the Last Stack

To restore the stack to its previous state:

- Press [↰][LAST STACK].

[↰][LAST STACK] restores the stack to the way it was before you executed the most recent command.

Displaying Objects for Viewing and Editing

You can't always see all of the objects on the stack—you can see only the beginning of large objects, and you can't see objects that have changed levels and scrolled off the display.

The HP 48 gives you a choice of *environments* for viewing and editing objects. An environment defines a particular display and keyboard behavior—it determines how you see and change the object.

To view or edit an object:

1. Depending on the location of the object and the desired environment, press the keys listed in the table below.
2. View or edit the object according to the rules of the environment.
3. Exit the environment:
   - To exit after viewing, press [ATTN].

- To save changes you've made, press ENTER.
- To discard changes you've made, press ATTN.

Depending on the location of an object, you have up to three ways to view or edit it.

Viewing or Editing an Object

Location of Object	Viewing/Editing Environment	Keystrokes to View or Edit
Level 1	Command line	◄ EDIT
	Best	▼
	Interactive Stack	▲
Level *n*	Command line	*n* ► VISIT
	Best	*n* ► ▼
	Interactive Stack	▲
Variable *name*	Command line	' *name* ► VISIT
	Best	' *name* ► ▼

The command line is the simplest viewing and editing environment:

- The EDIT menu is displayed, which provides operations that make it easier to edit large objects. (See "The Command Line and the EDIT Menu" below.)
- Real and complex numbers are displayed with full precision (standard format), regardless of the current display mode.
- Programs, lists, algebraics, units, directories, and matrices are formatted onto multiple lines.
- All the digits of binary numbers, all the characters in strings, and entire algebraic expressions are displayed.

The "best" editing environment is the one that's most appropriate based on the type of object. Algebraic objects and unit objects are copied into the EquationWriter environment (see "Viewing and Editing Objects with the EquationWriter Application" on page 16-23). Matrices are copied into the MatrixWriter environment (see "Viewing

3

and Editing Arrays" on page 20-6). All other object types are copied into the command line.

The Interactive Stack is an environment for viewing, editing, and manipulating all objects on the stack. See "The Interactive Stack" on page 3-9.

The Command Line and the EDIT Menu

If a command line is present, press (◀)(EDIT) to get the EDIT menu. Also, the EDIT menu is displayed whenever you perform a viewing or editing operation as described in the previous section.

Certain operations in the EDIT menu use the concept of a *word*—a series of characters between spaces or newlines. For example, pressing ←SKIP skips to the beginning of a *word*.

EDIT Menu Operations

Key	Description
(◀)(EDIT):	
←SKIP	Moves the cursor to the beginning of the current word.
SKIP→	Moves the cursor to the beginning of the next word.
←DEL	Deletes characters from the beginning of the word to the cursor.
DEL→	Deletes characters from the cursor to the end of the word.
(▶) ←DEL	Deletes characters from the beginning of the line to the cursor.
(▶) DEL→	Deletes all characters from the cursor to the end of the line.
INS	Switches the command-line entry mode between *Insert* mode (⁜ cursor) and *Replace* mode (█ cursor). A ■ in the menu label indicates Insert mode is active.
↑STK	Activates the Interactive Stack. (See the next topic, "The Interactive Stack.")

Example: Enter an algebraic object on the stack.

(⬅)(EQUATION) A (+) B (÷) C (y^x) 2 (ENTER)

```
1:                'A+B/C^2'
STD■ FIX SCI ENG SYM■ BEEP■
```

Edit the expression to make C^2 become C^3.

(⬅)(EDIT) SKIP→ (◀)(◀) (DEL) 3

```
'A+B/C^3◆
←SKIP SKIP→ ←DEL DEL→ INS■ ↑STK
```

Save the change, then view the edited equation in the EquationWriter environment.

(ENTER)
(▼)

$$A+\frac{B}{C^3}$$

Return to the stack.

(ATTN) (ATTN)

```
1:                'A+B/C^3'
STD■ FIX SCI ENG SYM■ BEEP■
```

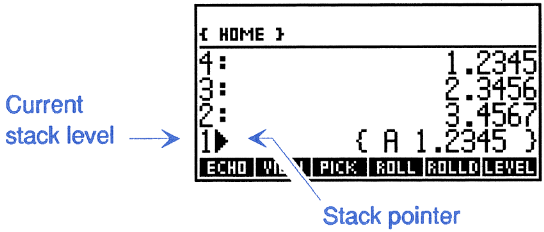
The Interactive Stack

The normal stack display is a "window" that shows level 1 and as many higher levels as will fit in the display. The *Interactive Stack* lets you:

- Move the window to see the rest of the stack.
- Move and copy objects to different levels.
- Copy the contents of any stack level to the command line.
- Delete objects from the stack.
- Edit stack objects.
- View stack objects in an appropriate environment.

The Interactive Stack is a special environment in which the keyboard is redefined for a specific set of stack-manipulation operations only. You must exit the Interactive Stack before you can execute any other calculator operations.

When you activate the Interactive Stack, the *stack pointer* turns on (pointing to the *current stack level*), the keyboard is redefined, and the Interactive-Stack menu is displayed.



To use the Interactive Stack:

1. Press (▲) (or press ↑STK in the EDIT menu) to activate the Interactive Stack.
2. Use the keys described in the following table to view or manipulate the stack.
3. Press (ATTN) (or (ENTER)) to leave the Interactive Stack and show the changed stack.
4. Optional: To cancel changes made to the stack in the Interactive Stack, press (↰)(LAST STACK).

If a command line is present when you select the Interactive Stack, just the ECHO key appears in the menu.

Interactive-Stack Operations

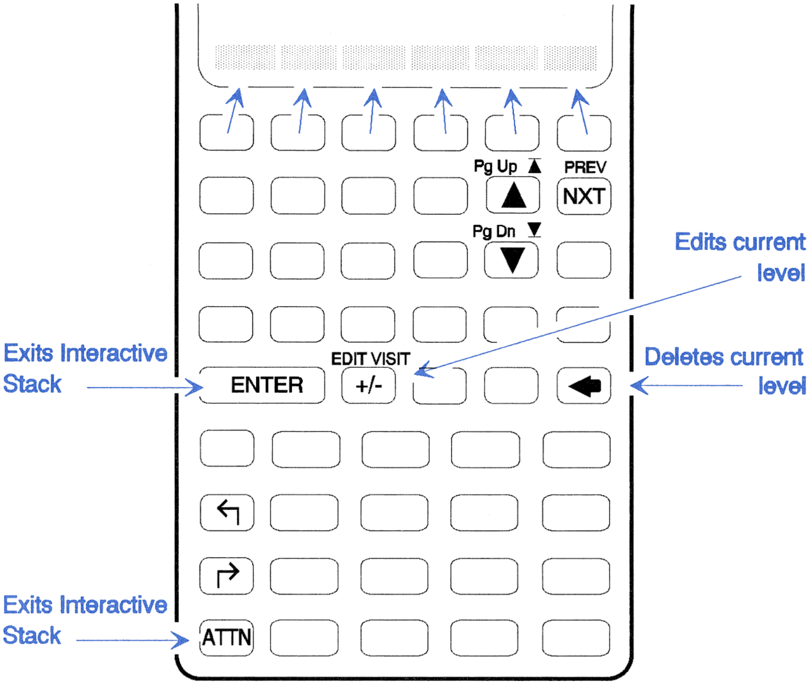
Key	Description
(▲):	
ECHO	Copies the contents of the current stack level into the command line at the cursor position.
VIEW	Views or edits the object in the current level using the "best" environment. Press (ENTER) when finished editing (or (ATTN) to abort).
(↱) VIEW	Views or edits the object specified by the name or stack-level number using the "best" environment. Press (ENTER) when finished editing (or (ATTN) to abort).
PICK	Copies the contents of the current level to level 1 (equivalent to *n* PICK).
ROLL	Moves the contents of the current level to level 1, and rolls (upwards) the portion of the stack beneath the current level (equivalent to *n* ROLL).
ROLLD	Moves the contents of level 1 to the current level, and rolls (downwards) the portion of the stack beneath the current level (equivalent to *n* ROLLD).
→LIST	Creates a list containing all the objects in levels 1 through the current stack level (equivalent to *n* →LIST).
DUPN	Duplicates levels 1 through the current stack level (equivalent to *n* DUPN). For example, if the pointer is at level 3, levels 1, 2, and 3 are copied to levels 4, 5, and 6.
DRPN	Drops levels 1 through the current stack level (equivalent to *n* DROPN).

Interactive-Stack Operations (continued)

Key	Description
KEEP	Clears all the stack levels above the current level.
LEVEL	Enters the current stack level number into level 1.
▲	Moves the stack pointer up one level. When prefixed with ↰, moves the stack pointer up four levels (↰ PgUp in the following keyboard illustration); when prefixed with ↱, moves the stack pointer to the top of the stack (↱ ⤒ in the following keyboard illustration).
▼	Moves the stack pointer down one level. When prefixed with ↰, moves the stack pointer down four levels (↰ PgDn in the following keyboard illustration); when prefixed with ↱, moves the stack pointer to the bottom of the stack (↱ ⤓ in the following keyboard illustration).
↰ EDIT	Copies the object in the current level into the command line for editing. Press ENTER when finished editing (or ATTN to abort).
↱ VISIT	Copies the object specified by the name or stack-level number into the command line for editing. Press ENTER when finished editing (or ATTN to abort).
◀	Deletes the object in the current level.
NXT	Selects the next page of Interactive-Stack operations.
ENTER	Exits the Interactive Stack.
ATTN	Exits the Interactive Stack.

Most of the operations in the menu have equivalent programmable commands—see "Other Stack Commands" on page 3-18.

The redefined keyboard for the Interactive Stack looks like this:



When you press (→) VIEW or (→)(VISIT) in the Interactive Stack, the object in the current level must be a real number or a variable name:

- A real number (integer part) specifies the object in the corresponding stack level.
- A variable name specifies the object stored there.

When you press VIEW or (→) VIEW, the object is copied into the "best" environment for viewing or editing. Algebraic objects and unit objects are copied into the EquationWriter environment (see "Viewing and Editing Objects with the EquationWriter Application" on page 16-23). Matrices are copied into the MatrixWriter environment (see "Viewing and Editing Arrays" on page 20-6). All other object types are copied into the command line. Press (ENTER) to return the edited object to its original stack level, or (ATTN) to end the session without change.

To copy an object from the stack into the command line:

1. Put the cursor in the command line where you want the object placed.
2. Press (⬅)(EDIT) ↑STK . (If the command line has only one line, you can press (▲) instead.)
3. Press (▲) and (▼) to move the Interactive Stack pointer to the desired object and press ECHO .
4. Press (ATTN) (or (ENTER)) to leave the Interactive Stack.

Example: Use the Interactive Stack to insert the number 1.2345 into the command line, creating the list { A 1.2345 }.

Put these numbers on the stack.

1.2345 (ENTER)
2.3456 (ENTER)
3.4567 (ENTER)

Start entering the list.

(⬅)({}) A

Select the Interactive Stack.

(▲)

Move the pointer to level 3, echo the object, and leave the Interactive Stack.

(▲) (▲) ECHO (ATTN)

Put the list on the stack.

(ENTER)

Using the Command Line

3

The command line is essentially a workspace for creating objects—a space where you enter and edit your input to the HP 48. The command line appears whenever you enter or edit text (except when you're using the EquationWriter or MatrixWriter applications).

Accumulating Data in the Command Line

You can key any number of characters into the command line, using up to half of the available memory. To enter more than one object in the command line, use spaces, newlines ([↱][↵]), or delimiters to separate objects. For example, you can key in 12 [SPC] 34 to enter two numbers—if you press [ENTER], they're entered onto the stack—or, if you press [+], they're entered and then added.

If you enter an @ character *not* inside a string in the command line, it and the adjacent text are treated as a "comment" and are stripped away when you press [ENTER]. See "Creating Programs on a Computer" on page 25-12.

When you type in the command line, characters are normally *inserted* at the cursor position—any trailing characters move to the right. The following keys are active when the command line is present.

Command Line Operations

Key	Description
◀ ▶	Move the cursor left and right in the command line. (↱◀ and ↱▶ move the cursor far left and far right.)
▲ ▼	If the command line has more than one line, move the cursor up and down one line. (↱▲ and ↱▼ move the cursor to the first and last line.) If the command line has only one line, ▲ selects the Interactive Stack, and ▼ displays the EDIT menu.
◆	Erases the character to the left of the cursor.
DEL	Deletes the character at the current cursor position.
↰ EDIT	Displays the EDIT menu, which contains additional editing operations.
↱ ENTRY	Changes the command-line entry mode to Program-entry mode or Algebraic/Program-entry mode, as described below.
ENTER	Processes the text in the command line—moves objects to the stack and executes commands.

Selecting Command-Line Entry Modes

Four command-line entry modes make it easier for you to key in various types of objects. Usually the entry mode you need is automatically activated for you.

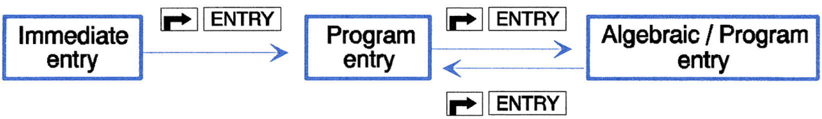
- **Immediate-Entry Mode.** (Activated automatically, indicated by no entry-mode annunciator.) Immediate-entry mode is the default mode. In Immediate-entry mode, the contents of the command line are entered and processed immediately when you press a function or command key (such as +, SIN, or STO).
- **Algebraic-Entry Mode.** (Activated when you press '. Indicated by the ALG annunciator.) Algebraic-entry mode is used primarily for keying in names and algebraic expressions for immediate use. In Algebraic-entry mode, command keys act as typing aids (for example, SIN types SIN()). Other commands *are* executed immediately (for example, STO or ↰ PURGE).

- **Program-Entry Mode.** (Activated when you press [↰][« »] or [↰][{ }]—indicated by the PRG annunciator.) Program-entry mode is used primarily for entering programs and lists. It's also used for command-line editing ([↰][EDIT] and [↱][VISIT]). In Program-entry mode, function keys and command keys act as typing aids (for example, [SIN] types SIN and [STO] types STO). Only non-programmable operations are executed when you press a key (for example, [ENTER], [VAR], or [↱][ENTRY]).
- **Algebraic/Program-Entry Mode.** (Activated when you press ['] while in Program-entry mode—indicated by the ALG *and* PRG annunciators.) Algebraic/Program-entry mode is used for keying algebraic objects into programs. In Algebraic/Program-entry mode, function and command keys behave as they do in Algebraic-entry mode (for example, [SIN] types SIN()). Pressing a command key (for example, [STO]) restores Program-entry mode.

To change entry modes manually:

- Press [↱][ENTRY].

Pressing [↱][ENTRY] switches from Immediate-entry to Program-entry mode, and between Program-entry and Algebraic/Program-entry modes.



[↱][ENTRY] allows you to accumulate commands in the command line for later execution. For example, you can manually invoke Program-entry mode to enter 4 5 + √ into the command line, and then press [ENTER] to calculate $\sqrt{4+5}$. [↱][ENTRY] also makes it easier to edit algebraic objects in programs.

Example: Calculate 12 − log(100) by including the LOG command in the command line. First, enter the command line.

12 [SPC] 100 [↱][ENTRY] [↱][LOG]

Process the command line to complete the calculation.

ENTER −

```
1:                 10
STD  FIX  SCI  ENG  SYM  BEEP
```

Recovering Previous Command Lines

The HP 48 automatically saves a copy of the four most recently executed command lines.

To retrieve a recent command line:

1. Press ⬅ LAST CMD (found over the 3 key).
2. If necessary, press ⬅ LAST CMD one or more times to retrieve an earlier saved command line.

Other Stack Commands

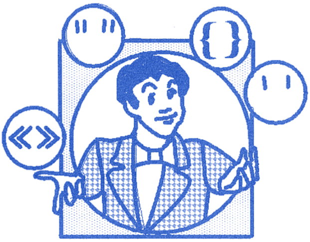
The following table describes additional commands from the PRG STK menu that are programmable and that manipulate the stack.

Command/Description	Example	
	Input	**Output**
DEPTH Returns the number of objects on the stack.	3: 2: 16 1: 'X1'	3: 16 2: 'X1' 1: 2
DROP2 Removes the objects in levels 1 and 2.	3: 12 2: 10 1: 8	3: 2: 1: 12
DROPN Removes the first $n + 1$ objects from the stack (n is in level 1).	4: 123 3: 456 2: 789 1: 2	4: 3: 2: 1: 123
DUP Duplicates the object in level 1.	3: 2: 232 1: 543	3: 232 2: 543 1: 543

Command/Description	Example	
	Input	Output
DUP2 Duplicates the objects in levels 1 and 2.	4: 3: 2: 'A' 1: (2,3)	4: 'A' 3: (2,3) 2: 'A' 1: (2,3)
DUPN Duplicates n objects on the stack, starting at level 2 (n is in level 1).	6: 5: 4: 123 3: 456 2: 789 1: 3	6: 123 5: 456 4: 789 3: 123 2: 456 1: 789
OVER Returns a copy of the object in level 2.	3: 2: 'AB' 1: 1234	3: 'AB' 2: 1234 1: 'AB'
PICK Returns a copy of the object in level $n + 1$ to level 1 (n is in level 1).	4: 123 3: 456 2: 789 1: 3	4: 123 3: 456 2: 789 1: 123
ROLL Moves object in level $n + 1$ to level 1 (n is in level 1).	5: 555 4: 444 3: 333 2: 222 1: 4	5: 4: 444 3: 333 2: 222 1: 555
ROLLD Rolls down a portion of the stack between level 2 and level $n + 1$ (n is in level 1).	6: 12 5: 34 4: 56 3: 78 2: 90 1: 4	6: 5: 12 4: 90 3: 34 2: 56 1: 78
ROT Rotates the first three objects on the stack (equivalent to `3 ROLL`).	3: 12 2: 34 1: 56	3: 34 2: 56 1: 12

4

Objects



The basic items of information the HP 48 uses are called *objects*. For example, a real number, a matrix, and a program are each an object. On the stack, an object occupies a single level—so you can enter any of them into one stack level—or store it in a variable.

The HP 48 can store and manipulate several *types* of objects:

Real Numbers	Programs	Directory Objects
Complex Numbers	Strings	Backup Objects
Binary Integers	Lists	Library Objects
Arrays	Graphics Objects	XLIB Names
Names	Tagged Objects	Built-In Functions
Algebraic Objects	Unit Objects	Built-In Commands

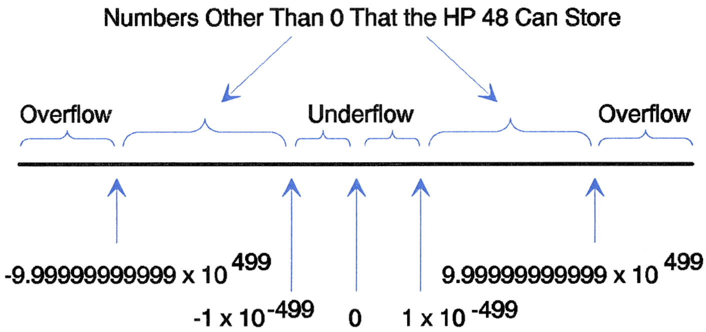
Many HP 48 operations are the same for all object types—for example, you use the same procedure to store a real number, matrix, or program. Other operations apply to only certain object types—for example, you can't take the square root of a program.

This chapter introduces the HP 48 object types, shows some examples of how you use different types of objects, and covers some commands that manipulate objects. Each object type is covered in more detail in another chapter.

4

Real Numbers

The numbers 12, -3.6, and 4.7E10 are examples of real numbers. The following illustration shows the range of real numbers the HP 48 can store.



Complex Numbers

A complex number is represented by a pair of real numbers delimited by parentheses. You can enter and display complex numbers in rectangular or polar form:

- **Rectangular form.** $x + iy$, displayed as (x,y).
- **Polar form.** ($re^{i\theta}$), displayed as (r,∠θ).

Complex numbers are also used to represent the coordinates of a point in two dimensions.

Example: Add the complex numbers $14 + 9i$ and $8 - 12i$.

If the R∠Z or R∠∠ annunciator is on, press [⮫] [POLAR] to set Rectangular coordinate mode (no coordinate annunciator).

Enter the complex numbers into levels 1 and 2. Use a space to separate the real and imaginary parts of each complex number.

[⮪] [( )] 14 [SPC] 9 [ENTER]
[⮪] [( )] 8 [SPC] 12 [+/-] [ENTER]

Add the two values.

(+)

```
1:                    (22,-3)
 STD ■ FIX  SCI  ENG  SYM ■ BEEP■
```

Complex numbers are covered in chapter 11.

Binary Integers

HP 48 binary integers are unsigned integers stored as a sequence of binary bits (rather than as decimal digits). They're delimited by a # character preceding the number and by an optional lowercase letter (h, d, o, or b) that identifies the current base. You can enter binary integers in hexadecimal, decimal, octal, or binary base. The binary base mode, set in the BASE menu (displayed by pressing (MTH) BASE), determines which base is active.

Example: Calculate $B17_{16} + 47_{8}$. Display the result in hexadecimal base.

Select hexadecimal base and enter the two values. Append the lowercase letter o ((α) (◀) O) to the octal value to specify its base.

(MTH) BASE HEX
(▶)(#) B17 (ENTER)
(▶)(#) 47o (ENTER)

```
2:                    # B17h
1:                     # 27h
 HEX ■ DEC  OCT  BIN  STWS RCWS
```

Add the two values.

(+)

```
1:                    # B3Eh
 HEX ■ DEC  OCT  BIN  STWS RCWS
```

Press DEC to return to decimal base.

Binary integers are covered in chapter 14.

Arrays

Arrays can be one-dimensional *vectors* or two-dimensional *matrices*. The delimiters for arrays are square brackets ([ ]). The HP 48 MatrixWriter application helps you enter and edit matrices.

Example: Multiply the following matrix and vector.

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

Enter the vector as [2 1 2]. Use spaces to separate the components.

(⬅) ([ ]) 2 (SPC) 1 (SPC) 2 (ENTER)

Now use the MatrixWriter application to enter the matrix. First, select the MatrixWriter application.

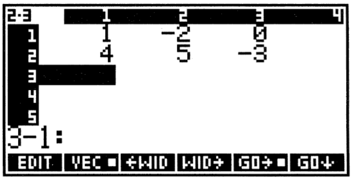
(➡) (MATRIX)

Enter the first row.

1 (ENTER)
2 (+/-) (ENTER)
0 (ENTER)

Start a new row and enter the three values. You can enter them one at a time, or you can enter them all at once by separating them with spaces.

(▼)
4 (SPC) 5 (SPC) 3 (+/-) (ENTER)



Now enter the matrix into level 1.

ENTER

```
2:              [ 2 1 2 ]
1: [[ 1 -2 0 ]
    [ 4 5 -3 ]]
 HEX  DEC■ OCT  BIN  STWS RCWS
```

To do the multiplication, the matrix must be in level 2 and vector must be in level 1, so swap the levels. (Pressing ▶ when no command line is present is the same as pressing ↰ SWAP.)

▶

```
2: [[ 1 -2 0 ]  [ 4 5…
1:              [ 2 1 2 ]
 HEX  DEC■ OCT  BIN  STWS RCWS
```

Multiply them.

×

```
1:                [ 0 7 ]
 HEX  DEC■ OCT  BIN  STWS RCWS
```

Arrays are covered in chapter 20.

Names

Names are used to identify variables. If you want to put a name on the stack without evaluating it, enclose it between ' (tick) characters.

Example: Enter the names *A1* and *B1* and multiply them.

Enter the names on the stack. The ALG annunciator comes on when you press '.

' A1 ENTER
' B1 ENTER

```
2:                   'A1'
1:                   'B1'
 HEX  DEC■ OCT  BIN  STWS RCWS
```

Multiply the two names.

×

```
1:                'A1*B1'
 HEX  DEC■ OCT  BIN  STWS RCWS
```

The result is an algebraic expression containing the two names.

Variable names are covered in chapter 6.

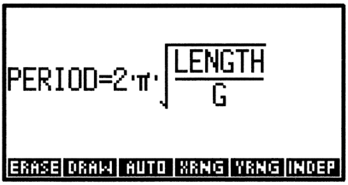
Algebraic Objects

Algebraic objects, like names, are delimited by two ' marks. Algebraics represent mathematical expressions and, on the stack, have a conventional "computer form" like these two examples:

```
'PERIOD=2*π*√(LENGTH/G)'

'∂X(2*X^3+COS(X))'
```

The EquationWriter application helps you enter and manipulate algebraic objects by displaying them as they might appear printed in a book. For example, this is how the PERIOD equation above would look in the EquationWriter environment:



Algebraic objects, often referred to as *algebraics* in this manual, are covered in chapter 8. The EquationWriter application is covered in chapter 16.

Programs

Programs are sequences of commands and other objects enclosed by the delimiters « and ». For example, given a real number argument in level 1 representing the radius, this program computes the area of a circle:

```
« SQ π →NUM * »
```

The delimiters prevent the commands from being executed as you enter them. Instead, they're executed later when you evaluate the program object.

Programming is covered in part 4 (chapters 25 through 31).

Strings

Strings are sequences of characters, usually used to represent text in programs. They're delimited by quotation marks. For example, you can enter the string "Minor of a Matrix" onto the stack and then print it.

A *counted string* is an alternate string form in which the number of characters is specified. Counted strings are prefaced with C$ *n*, where *n* is a real integer. C$ designates that the string is a counted string, and *n* specifies the number of characters to be gathered into the string. For example, entering C$ 7 ABC DEF GHI from the command line results in the creation of the string "ABC DEF". The leftover GHI is entered as a name, just like it would be entered if it were on the stack by itself.

Another form of the counted string prefaces the string with C$ $. This form specifies that all characters remaining on the command line are put into the string.

Lists

Lists are sequences of objects grouped together, delimited by braces—for example, { X 0 1 }. Lists allow you to combine objects so they can be manipulated as one object.

Graphics Objects

Graphics objects encode the data for HP 48 "pictures," including plots of mathematical data, custom graphical images, and representations of the stack display itself. They're created by certain plotting commands, and they're viewed in the Graphics environment. They can also be put on the stack and stored in variables. On the stack, a graphics object is displayed as

Graphic *n* × *m*

where n and m are the width and height in *pixels*. (A pixel is one dot in the display.)

Creating and manipulating graphics objects is covered in chapters 18 and 19.

Tagged Objects

A tagged object consists of any object combined with a tag that labels that object. Tagged objects are keyed in as

:*tag*: *object*

The colons delimit the tag. When a tagged object is displayed on the stack, the leading colon is dropped for readability.

Example: Enter the numbers 5 and 9 with tags "B1" and "B2", and then calculate the product.

Enter the tagged objects.

[↱] [::] B1 [▶] 5 [ENTER]
[↱] [::] B2 [▶] 9 [ENTER]

Calculate the product.

[×]

The tags were ignored by [×].

Tagged objects are particularly useful for labeling the contents of variables and program output (see "Labeling Output with Tags" on page 29-14).

Unit Objects

A unit object consists of a real number combined with measurement units. The underscore character (_) separates the units from the number—for example, `2_m` and `26.7_kg*m^2/s^2`.

Example: Calculate $\frac{50.8 \text{ ft/s}}{2.5 \text{ s}}$.

Enter the unit objects 50.8 ft/s and 2.5 s.

50.8 [⬅] [UNITS] LENG FT
[⬅] [UNITS] TIME [➦] S
2.5 S

```
2:           50.8_ft/s
1:               2.5_s
 YR  D  H  MIN  S  HZ
```

Divide the two values.

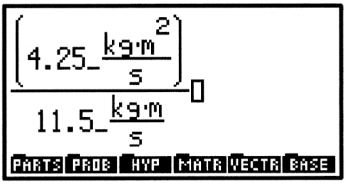
[÷]

```
1:        20.32_ft/s^2
 YR  D  H  MIN  S  HZ
```

Algebraic objects can contain unit objects, such as

`'(4.25_kg*m^2/s)/(11.5_kg*m/s)'`

When units are included in algebraic objects, the EquationWriter application helps you enter, edit, and manipulate them. Here is the same expression as it's displayed by the EquationWriter application:



Unit objects are covered in chapter 13.

Directory Objects

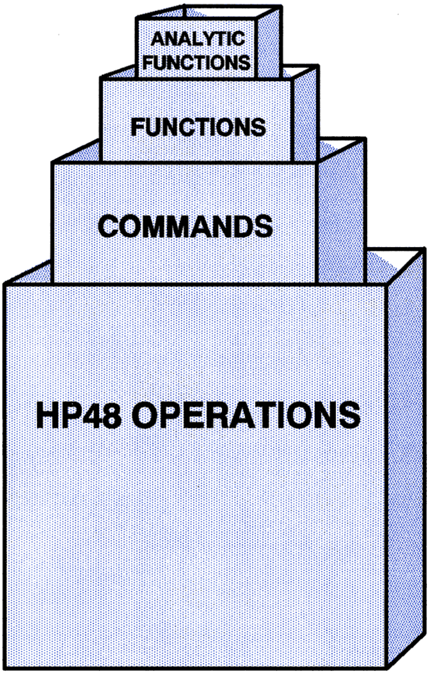
The HP 48 uses directory objects to set up a hierarchical directory structure for data you store. Directory objects are covered in chapter 7.

Built-In Functions and Commands

Built-in *functions* and built-in *commands* are subsets of HP 48 *operations*. An operation is any action the calculator can perform. (Every time you press a key, you execute an operation.) Later on, it will be helpful to know if an operation can be included in a program, if it can be included in an algebraic object, and if it has an inverse or derivative. Therefore, operations are classified by these categories throughout this manual:

- **Operation.** Any action built into the calculator represented by a name or key.
- **Command.** Any programmable operation.
- **Function.** Any command that can be included in algebraic objects.
- **Analytic function.** Any function for which the HP 48 provides an inverse and derivative.

Analytic functions are a subset of functions—functions are a subset of commands—and commands are a subset of operations. 4



SIN, for example, is an analytic function—it has an inverse and derivative, can be included in an algebraic object, and is programmable. SWAP (the command to swap stack levels 1 and 2), however, is just a command—it can be included in a program, but it can't go in an algebraic and has no derivative or inverse.

The operation index in appendix G tells you how each operation is classified. Also, throughout the manual, HP 48 activities are referred to as operations, commands, functions, or analytic functions where appropriate.

Built-in function and built-in command objects describe the HP 48 command set. You can think of them as built-in program objects. (Operations that aren't commands are not objects—you can't include them in programs.)

Additional Object Types

Three object types involve operations with plug-in cards (covered in chapter 34):

- Backup objects. They are created when you store objects in a plug-in memory card.
- Library objects. A library is a directory of commands and operations that are *not* built into the calculator. Libraries can be provided by a plug-in application card, or they can exist in built-in or plug-in RAM.
- XLIB names. These are objects provided by plug-in application cards.

Manipulating Objects

You can use several commands for assembling, disassembling, and modifying portions of objects. These commands (except +) are located in the PRG OBJ (program object) menu ((PRG) OBJ).

Command/Description	Example	
	Input	Output
+ (+) Combines two strings or lists, or adds an object to a string or list.	2: 'A' 1: { 2 3 } 2: "ABC" 1: "DE"	2: 1: { A 2 3 } 2: 1: "ABCDE"

Command/Description	Example	
	Input	Output
→ARRY (→ARR) Stack to array; combines real or complex numbers into an n-element rectangular vector or a matrix of dimensions n by m. (n or { n m } is in level 1.)	3: 8 2: 9 1: 2 7: 1 6: 2 5: 3 4: 4 3: 5 2: 6 1: { 3 2 }	3: 2: 1: [8 9] 5: 4: 3: 2: 1: [[1 2] [3 4] [5 6]]
CHR Character; returns the character corresponding to the character code. (See the character set in appendix C.)	1: 140	1: "α"
C→R Complex to real; separates a complex number (or complex array) into two real numbers (or real arrays) representing the real part and the imaginary part.	2: 1: (2,3)	2: 2 1: 3
DTAG Delete tag; removes the tag from a tagged object.	1: A:123	1: 123
EQ→ Equation to stack; splits an equation into its left and right sides.	2: 1: 'A=B+C' 2: 1: 'B+C'	2: 'A' 1: 'B+C' 2: 'B+C' 1: 0

Command/Description	Example	
	Input	**Output**

GET Gets the *n*th (in level 1) element of a vector, matrix, or list, or the { *n* *m* } element of a matrix.	`2: [4 5 6]`
	`1: 2`
	
	`2: [[4 5 6]`
	`[7 8 9]]`
	`1: 4`
	
	`2: [[4 5 6]`
	`[7 8 9]]`
	`1: { 2 1 }`
	
	`2: { A B C }`
	`1: 2`
		`2:`
		`1: 5`
		
		`2:`
		`1: 7`
		
		`2:`
		`1: 7`
		
		`2:`
		`1: 'B'`

GETI Get and increment; same as GET, except also returns the vector, matrix, or list to level 3 and the number of the next element to level 2.	`3:`
	`2: [4 5 6]`
	`1: 2`
	
	`3:`
	`2: [[4 5 6]`
	`[7 8 9]]`
	`1: 4`
	
	`3:`
	`2: [[4 5 6]`
	`[7 8 9]]`
	`1: { 2 2 }`
		`3: [4 5 6]`
		`2: 3`
		`1: 5`
		
		`3: [[4 5 6]`
		`[7 8 9]]`
		`2: 5`
		`1: 7`
		
		`3: [[4 5 6]`
		`[7 8 9]]`
		`2: { 2 3 }`
		`1: 8`

→LIST Stack to list; creates a list containing *n* (in level 1) objects.	`3: 'X+Z'`
	`2: 'X'`
	`1: 2`
		`3:`
		`2:`
		`1: {'X+Z' X}`

NUM Returns the character code corresponding to the character.	`1: "A"`	`1: 65`

Command/Description	Example	
	Input	Output
OBJ→ Object to stack; separates a complex number, array, or list into its elements (same as C→R, ARRY→, and LIST→); for arrays and lists, also returns the number of elements or dimensions to level 1. For strings, removes the string delimiters and executes the contents as a command line (same as STR→). For algebraics, separates the outermost function and its arguments. For units, separates the number and the unit expression. For tagged objects, separates the tag and the object.	2: 1: (4,5)	2: 4 1: 5
	3: 2: 1: [8 9]	3: 8 2: 9 1: { 2 }
	4: 3: 2: 1: [[1 2] [5 6]]	5: 1 4: 2 3: 5 2: 6 1: { 2 2 }
	4: 3: 2: 1: {1 2 Y}	4: 1 3: 2 2: 'Y' 1: 3
	1:"5 SQ 2 *"	1: 50
	4: 3: 2: 1: 'A + B'	4: 'A' 3: 'B' 2: 2 1: +

Command/Description	Example: Input	Example: Output
POS Position of an object in a list, or position of one string within another.	`2: { A 3 C }` `1: 'C'`	`2:` `1: 3`
	`2: "ABCDEFG"` `1: "DE"`	`2:` `1: 4`

PUT Replaces the *n*th element of a vector, matrix, or list, or the { *n* *m* } element of a matrix, with the contents of level 1. (*n* or { *n* *m* } is in level 2.)	`3: [4 5 6]`
	`2: 2`
	`1: 7`
		`3:`
		`2:`
		`1: [4 7 6]`

	`3: [[4 5 6]`
	`[7 8 9]]`
	`2: 4`
	`1: 2`
		`3:`
		`2:`
		`1: [[4 5 6]`
		`[2 8 9]]`

	`3: { 7 8 9 }`
	`2: 2`
	`1: 'A'`
		`3:`
		`2:`
		`1: { 7 A 9 }`

PUTI Put and increment; same as PUT, except also returns the list or array to level 2 and the number of the next element to level 1.	`3: [4 5 6]`
	`2: 2`
	`1: 7`
		`3:`
		`2: [4 7 6]`
		`1: 3`

	`3: [[4 5 6]`
	`[7 8 9]]`
	`2: 4`
	`1: 2`
		`3:`
		`2: [[4 5 6]`
		`[2 8 9]]`
		`1: 5`

	`3: { 7 8 9 }`
	`2: 2`
	`1: 'A'`
		`3:`
		`2: {7 A 9}`
		`1: 3`

Command/Description	Example	
	Input	Output

REPL Replace; replaces a portion of a list or string in level 3. Takes the replacement object from level 1 and the position in the list or string to start the replacement from level 2. (How REPL works with graphics objects is described under "Using Stack Commands for Graphics Objects" on page 19-27.)	`3: {A B C D}`
	`2: 2`
	`1: {F G}`
	
	`3: {A B C}`
	`2: 3`
	`1: {F G}`
	
	`3: {A B}`
	`2: 10`
	`1: {F G}`
		`3:`
		`2:`
		`1: {A F G D}`
		
		`3:`
		`2:`
		`1: {A B F G}`
		
		`3:`
		`2:`
		`1: {A B F G}`

R→C Real to complex; combines two real numbers or arrays into a complex number or complex array.	`2: -7` `1: -2`	`2:` `1: (-7,-2)`

SIZE Number of elements in a list; number of characters in a string; dimension of an array; and size of a graphics object.	`1: {UX 2 Y}`
	
	`1: "ABCDEFG"`
	
	`1: [[4 5 6]`
	`[7 8 9]]`
	
	`GRAPHIC 6x12`
		`1: 3`
		
		`1: 7`
		
		`1: { 2 3 }`
		
		`2: # 6d`
		`1: #12d`

→STR Object to string; converts an object to a string.	`1: 'A+B'`	`1: "'A+B'"`

Command/Description	Example	
	Input	Output

SUB Subset of a list or string. The positions of the beginning and ending elements are in levels 2 and 1.	`3: { A B C }`
	`2: 2`
	`1: 3`
	
	`3: "ABCDEFG"`
	`2: 3`
	`1: 5`
		`3:`
		`2:`
		`1: { B C }`
		
		`3:`
		`2:`
		`1: "CDE"`

→TAG Stack to tag; combines two objects to form a tagged object.	`2: 123` `1: 'Value'`	`2:` `1:Value: 123`
→UNIT Stack to unit; assembles a scalar from level 2 and a unit expression from level 1 to form a unit object.	`2: 85` `1: 17_m`	`2:` `1: 85_m`

GET, GETI, PUT, and PUTI allow name arguments in place of the array argument. For example, evaluating `'A1' 2 GET` returns the second element of *A1*; evaluating `'A2' 2 "ABC" PUT` replaces the second element in *A2* with "ABC".

Determining Object Types

There are 20 types of objects used in the HP 48. Each object type is represented by an integer.

Object Type Numbers

Object	TYPE Number	Object	TYPE Number
Real number	0	Binary integer	10
Complex number	1	Graphics object	11
String	2	Tagged object	12
Real array	3	Unit object	13
Complex array	4	XLIB name	14
List	5	Directory	15
Global name	6	Library	16
Local name	7	Backup object	17
Program	8	Built-in function	18
Algebraic object	9	Built-in command	19

To find the type of object in level 1:

- Press (PRG) OBJ (NXT) TYPE (the TYPE command).

To find the type of object stored in a variable:

1. Press (') and enter the name of the variable.
2. Press (PRG) OBJ (NXT) VTYPE (the VTYPE command).

The TYPE and VTYPE commands return a number representing the type of object. VTYPE returns −1 if the variable doesn't exist.

Separating Variable Names by Object Type

To get a list of variables containing one type of object:

1. Enter the TYPE number for the object type you want.
2. Press (←)(MEMORY) (NXT) TVARS (the TVARS command).

The TVARS command returns a list containing the names of variables in the current directory containing that object type. For example, pressing TVARS with 8 in level 1 returns a list of all the names of variables containing programs. If no variables contain that object type, TVARS returns an empty list to the stack.

Evaluating Objects

Evaluation is the fundamental calculator operation for prodding an object into action. Evaluation is often implicit in calculator operations—it happens when commands are executed, programs are run, etc.

To evaluate an object in level 1:

- Press [EVAL] (the EVAL command).

The result of evaluating an object can be a sequence of subsequent actions, which can include further evaluations. The following table describes the effect of evaluating different types of objects.

Obj. Type	Effect of Evaluation
Local Name	Recalls the contents of the variable. If appropriate, the contents can then be explicitly evaluated using the EVAL command.
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 quoted name to the stack.
Program	Enters each object in the program: ■ Names are evaluated, unless delimited by tick marks ([']). ■ Commands are executed. ■ Other objects are put on the stack.

Obj. Type	Effect of Evaluation
List	Enters each object in the list: ■ Names are evaluated. ■ Programs are evaluated. ■ Commands are executed. ■ Other objects are put on the stack.
Algebraic	Enters each object in the algebraic: ■ Names are evaluated. ■ Commands are executed. ■ Other objects are put on the stack.
Other Objects	Puts the object on the stack.

Example: Suppose you created two global variables:

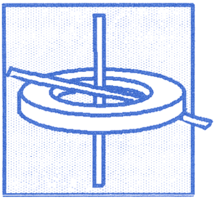
- *TWOPI* contains the real number 6.28318530718.
- *CIRCUM* contains the program « TWOPI * ».

The label CIRCU in the VAR menu represents *CIRCUM*. When you press CIRCU, here's what happens:

1. The name *CIRCUM* is evaluated.
2. The program stored in the variable *CIRCUM* is evaluated.
3. The name *TWOPI* (the first object in the program) is evaluated.
4. The real number stored in the variable *TWOPI* is returned to the stack.
5. The command $*$ (multiply) is executed.

5

Calculator Memory



Every operation you perform with your HP 48 requires memory. This chapter describes the types of memory, shows how to find out about memory usage, and tells how to respond to low-memory conditions.

Types of Memory

The HP 48 has two types of memory:

- **Read-only memory (ROM).** Memory that's dedicated to specific operations and cannot be altered. The HP 48 has 256 KB (kilobytes) of built-in ROM, which contains its command set. Except for the HP 48S model, you can expand the amount of ROM by installing plug-in application cards, which are described in chapter 34, "Using Plug-in Cards and Libraries."
- **Random-access memory (RAM).** Memory that you can change. You can store data into RAM, modify its contents, and purge data. The HP 48 contains 32 KB of built-in RAM. Except for the HP 48S model, you can increase the amount of RAM by adding memory cards, which are described in chapter 34.

RAM is also called *user memory* because it's memory that you (the user) have access to. You use or manipulate user memory when you enter an object on the stack, save an object in a variable, delete a variable, create an equation or matrix, run a program, etc. In addition, the HP 48 does some system cleanup from time to time to free memory for current operations.

The next two chapters, "Variables and the VAR Menu" and "Directories," cover the organization and management of user memory.

Finding Out about Memory Usage

To find out how much memory is available:

- Press (⬅)(MEMORY) MEM (the MEM command).

The MEM command returns the number of bytes of unused user memory. For exmaple, an empty memory for the HP 48SX should show about 30000 bytes of available memory (with no RAM cards installed).

To get the memory size and checksum of the object in level 1:

- Press (⬅)(MEMORY) BYTES (the BYTES command).

The BYTES command returns:

- **Level 2: Checksum.** The checksum is a binary integer specific to the object. You can use checksums to ensure that you've keyed in a large object (for example, a program or matrix) properly by comparing the checksum of the listing with the checksum you get after you've keyed it in. (Most programs in part 4 of this manual have a checksum at the end of the listing to help you verify that you've keyed in the program correctly.)
- **Level 1: Number of bytes.** The amount of memory in bytes the object takes up. If the object is a variable name, the memory used by the variable's name *and* contents is returned. If it's a built-in object, 2.5 bytes is returned.

Additional memory commands are covered in chapter 6, "Variables and the VAR Menu," and in chapter 7, "Directories."

Saving and Restoring the Stack

Certain HP 48 operations clear the stack—but you may want to keep the data stored there. You can save the contents of the stack in a variable, then restore the stack later.

To save the stack in a variable:

1. Press PRG STK DEPTH to get the size of the stack.
2. Press PRG OBJ →LIST to combine the stack into a list object.
3. Press ', type a name to use for storing the stack data, and press STO.

To restore the stack from a variable:

1. Optional: Press ↱ CLR to clear the current stack contents.
2. Press VAR and the menu key for the name you used for storing the stack data.
3. Press PRG OBJ OBJ→ to expand the list.
4. Press ◆ to drop the number of restored objects.

If you don't clear the stack, the current contents move up to levels above the restored contents.

Clearing All Memory

Clearing memory *erases all information you've stored* and resets all modes to their default settings. Therefore, you probably won't do this very often, or at least not without careful forethought.

5

To clear all memory:

1. Press and hold down these three keys simultaneously: ON, the leftmost menu key (A), and the rightmost menu key (F).
2. Release the two menu keys, then release ON. The calculator beeps and displays the message `Try To Recover Memory?`.
3. Press NO. The HP 48 beeps and displays `Memory Clear`.

If necessary, you can cancel the clearing operation before releasing ON by *continuing to hold down* ON as you press the second menu key from the left (B). You can also answer YES to the `Try To Recover Memory?` prompt—however, the calculator may not be able to recover all memory at that point. You probably would lose at least your stack, alarms, and user-key assignments.

Responding to Low-Memory Conditions

HP 48 operations share memory with the objects you create. Normal calculator operation becomes slow or fails if user memory is sufficiently full. When a low-memory condition occurs, the HP 48 returns one of a series of low memory warnings. These messages are described below in order of increasing severity.

"No Room for Last Stack". If there's not enough memory to save a copy of the current stack, the message `No Room for Last Stack` is displayed when ENTER is executed. Also, the LAST STACK operation (◀ LAST STACK) is disabled.

This condition indicates that user memory is getting full. You should make more room by deleting unnecessary objects from the stack or deleting unnecessary variables.

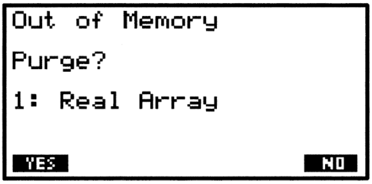
"Insufficient Memory". If there isn't enough memory to complete execution of an operation, `Insufficient Memory` is displayed. If the LAST ARG operation (▶ LAST ARG) is enabled (flag −55 is clear), the original arguments are restored to the stack. If LAST ARG is disabled (flag −55 is set), the arguments are lost.

Delete unnecessary objects from the stack or delete unnecessary variables.

"No Room to Show Stack". The HP 48 may complete all pending operations and then not have enough free memory to display the stack. In this case the calculator displays No Room To Show Stack in the top line of the display. Those lines of the display that would normally display stack objects, now show those objects only by type, for example, Real Number, Algebraic, and so on.

The amount of memory required to display a stack object varies with the object type. If there's no room to show the stack, delete unnecessary objects from the stack or delete unnecessary variables—or store a stack object in a variable so that it doesn't have to be displayed.

"Out of Memory". In the extreme case of low memory, there is insufficient memory for the calculator to do anything—display the stack, show menu labels, execute a command, etc. In this situation you *must* clear some memory before continuing. A special Out of Memory procedure is activated, which starts with the following display.



To respond to the "Out Of Memory" prompts:

- To delete the indicated object, press YES.
- To keep the indicated object, press NO.
- To stop the procedure and see if the condition is fixed, press (ATTN)

When the procedure starts, the display asks whether or not you want to purge the object (described by object type) in level 1—a real array in the example above. If you delete it, the choice is repeated for the new level 1 object. The succession of stack objects continues until the stack is empty or you press NO. Then, the prompt to delete level 1 is replaced by a prompt to discard the contents of LAST CMD ((⬅)(LAST CMD))—and then to delete other items in this order:

1. Stack level 1 (repeated).
2. The contents of LAST CMD.

3. The contents of LAST STACK (if active).
4. The contents of LAST ARG (if active).
5. The variable *PICT* (if present).
6. Any user-key assignments.
7. Any alarms.
8. The entire stack (unless already empty)
9. Each global variable by name.
10. Each port 0 object by tagged name.

Note

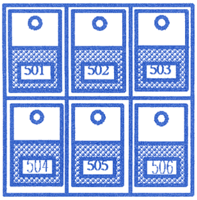
It is possible for the purge sequence to begin with the command line and *then* cycle through the stack, the contents of LAST CMD, etc. If you answer NO to the purge prompt for the command line, you'll be returned to the command line when you terminate the Out of Memory procedure.

The prompt for variables (prompt 9 above) starts with the newest object in the *HOME* directory and then proceeds with successively older objects. If the variable to be purged is an empty directory, YES purges it. If the directory is not empty, YES causes the variable-purge sequence to cycle through the variables (from newest to oldest) in that directory.

Whenever you like, you can try to terminate the Out of Memory procedure by pressing ATTN. If sufficient memory is available, the calculator returns to the normal display; otherwise, the calculator beeps and continues with the purge sequence. After cycling once through the choices, the HP 48 attempts to return to normal operation. If there still is not enough free memory, the procedure starts over with the sequence of choices to purge.

6

Variables and the VAR Menu



A *variable* is a named storage location that contains an object. Variables let you store and retrieve information using meaningful names. For example, you can store the acceleration of gravity, 9.81 m/s^2, into a variable named *G* and then use the name to refer to the variable's contents.

The HP 48 uses two types of variables:

- **Global variables.** Common variables that remain in memory until you purge them.
- **Local variables.** Temporary variables created by programs. They exist only while a portion of the program is being executed and can't be used outside the program.

This chapter covers only global variables. Local variables are covered under "Using Local Variables" on page 25-13.

Naming Variables

Variable names can contain up to 127 characters, and can contain letters, digits, and most other characters. If a name is too long to fit in a menu label, only the beginning of the name is shown.

Uppercase and lowercase letters are *not* equivalent—even though they appear the same in menu labels.

The following characters can't be included in names:

- Characters that separate objects: space, period, comma, @, and object delimiters # [ ] " ' { } ( ) « » : _.
- Mathematical function symbols: + − * / ^ √ = < > ≤ ≥ ≠ ∂ ∫ !.

Names can't begin with a digit. You can't use command names (for example, SIN, i, or π) as variable names. Also, PICT is a special name used by the HP 48 to contain the current graphics object and can't be used as a variable name. Certain names are legal variable names, but are used by the HP 48 for specific purposes. You can use these names, but remember that certain commands use them as implicit arguments—if you alter their contents, those commands may not execute properly. These variables are called *reserved variables*:

- *EQ* refers to the current equation used by HP Solve and Plot applications.
- *CST* contains data for custom menus.
- *ΣDAT* contains the current statistical matrix.
- *ALRMDAT* contains the data for an alarm being built or edited.
- *ΣPAR* contains a list of parameters used by STAT commands.
- *PPAR* contains a list of parameters used by PLOT commands.
- *PRTPAR* contains a list of parameters used by PRINT commands.
- *IOPAR* contains a list of parameters used by IO commands.
- *s1*, *s2*, ... , are created by ISOL and QUAD to represent arbitrary signs obtained in symbolic solutions.
- *n1*, *n2*, ... , are created by ISOL to represent arbitrary integers obtained in symbolic solutions.
- Names beginning with "der" refer to user-defined derivatives.

Creating Variables

To create a variable by storing an object:

1. Enter the object to store.
2. Enter the name of the variable. (Press ['] and type the name.)
3. Press [STO] (the STO command).

The STO command removes the object and name from the stack and stores the object in a variable with that name. If the variable doesn't exist, it's created in the current directory. (Directories are covered in chapter 7. If you haven't created any directories, all your variables are created in the *HOME* directory.) If the variable does exist, the new object replaces the old object.

You can store any object type in a variable.

To get the VAR menu of variables:

- Press [VAR].

The VAR menu contains a menu key for each variable in the current directory. You can use variable keys to type variable names and to access the contents of variables. See "Using the VAR Menu and REVIEW Catalog" on page 6-7.

Example: Create the variable *VCT1* containing the vector [1 2 3].

Enter the vector [1 2 3].

[⬅][[]] 1 [SPC] 2 [SPC] 3 [ENTER]

```
1:                [ 1 2 3 ]
 YR | D | H | MIN | S | HZ
```

Key in the variable name and press [STO]. If it's not displayed, display the VAR menu to see the new variable.

['] VCT1 [STO]
[VAR]

```
 VCT1 | DIFF | KE | X | TRAY | A1
```

To create a variable from a symbolic definition:

1. Enter an equation with a name on the left side and a symbolic definition on the right side.
2. Press [⬅][DEF] (the DEFINE command).

The DEFINE command can create variables from equations. If in level 1 you have an equation with a valid name as its left side, executing DEFINE stores the expression on the right side of the equation in the name on the left.

Example: Use DEFINE to store 6 in the variable *A*. Press `'A=6'` [⬅][DEF].

If flag −3 is clear (its default state), DEFINE stores the expression without evaluation. If you've set flag −3, the expression to be stored is evaluated to a number, if possible, before it's stored. For example, the keystrokes `'A=10+10'` [⬅][DEF] create variable *A* and store in it `'10+10'` if flag −3 is clear and `20` if flag −3 is set.

6

Using the Contents of Variables

After you've created a variable, you can access its contents two ways:

- *Evaluate* the variable's name. (This is the most common way of using variables.)
- *Recall* the variable's contents.

Evaluating Variable Names

Evaluating a variable name calls the object stored in the variable:

- **Name.** The name is evaluated (calling *its* object).
- **Program.** The program runs.
- **Directory.** The directory becomes the current directory.
- **Other Object.** A copy of the object is returned to the stack.

To evaluate a variable name:

- Press the variable's key in the VAR menu.

  or
- Enter the variable name (*not* inside tick marks) and press [ENTER].

For example, [G] evaluates *G*, and G [ENTER] evaluates *G*.

Example: Create three variables—*A* containing 2, *B* containing 5, and *ALG* containing the expression 'A+B'. Then evaluate them from the VAR menu.

Display the VAR menu and create the variables.

[VAR]
2 [ENTER] ['] A [STO]
5 [ENTER] ['] B [STO]
['] A [+] B [ENTER] ['] ALG [STO]

Evaluate *ALG*, *B*, and *A*. The contents of the variables are put on the stack.

[ALG]
[B]
[A]

Recalling the Contents of Variables

Recalling a variable puts a copy of its contents on the stack—nothing is evaluated.

To recall the contents of a variable:

- Press [➡] and the variable's key in the VAR menu.

  or
- Enter the name of the variable (inside tick marks), then press [➡] [RCL] (the RCL command).

Because recalling requires more keystrokes than evaluating a variable name, recalling is used primarily to get a copy of a variable containing a program, directory, or name. (For other object types, just evaluate the name.)

Example: Store a program in *ADD2*, then recall it.

Enter the program, then store it.

[⬅] [« »] [+] [+] [ENTER]
['] ADD2 [STO]

```
ADD2 | ALG | B | A | VCT1 | DIFF
```

Recall the program stored in *ADD2*.

[➡] ADD2

```
1:                    « + + »
ADD2 | ALG | B | A | VCT1 | DIFF
```

Changing the Contents of Variables

To change the contents of a variable:

- Enter the new contents in level 1, then press [⬅] and the variable's key in the VAR menu.

  or
- Enter the variable name in level 1 (inside tick marks), then press [➡] [VISIT], edit the contents, and then press [ENTER].

  or
- Enter the new contents in level 2 and the variable name in level 1 (inside tick marks), then press [STO] (the STO command).

Example: Change the contents of *ADD2* from « + + » to « 2 + ».

Enter the new contents.

(⬅) («») 2 (+) (ENTER)

Store the new contents in *ADD2*.

(⬅) ADD2

Using Quoted and Unquoted Variable Names

The ' delimiter is very important when you're entering a variable name. It determines whether the name is evaluated or not when you press (ENTER). If the ' delimiter is present, evaluation is prevented—if there's no delimiter, the unquoted name is automatically evaluated.

To enter a variable name:

- To put the name on the stack, press ('), then type the name or press its VAR menu key.
- To automatically evaluate the name, type the name or press its VAR menu key (*without* pressing (')).

Commands such as STO, RCL, and PURGE take a quoted variable name as an argument from the stack.

When you evaluate an unquoted variable name, the action taken depends on the object type of the contents. (See the earlier topic, "Evaluating Variable Names.")

If you execute an unquoted name that doesn't exist (it's a *formal variable*, no variable has been created with that name), the name is put on the stack *with quotes*. The ability to use names without having to create variables enables you to do symbolic math with the HP 48.

Using the VAR Menu and REVIEW Catalog

The VAR menu ((VAR)) contains a label for each global variable you've created in the current directory.

To use the VAR menu:

- To evaluate a variable name, press its menu key.
- To recall a variable's contents, press (↱) and the menu key.
- To change a variable's contents, enter the new contents on the stack, then press (↰) and the menu key.
- To type a variable name when the command line is in Algebraic- or Program-entry mode, press the menu key.
- To display the REVIEW catalog of variables, press (↰)(REVIEW). (Press (ATTN) to return to the stack display.)

The REVIEW catalog shows the full names and contents of variables on the current page of the VAR menu. The next keystroke you make cancels the review, redisplays the stack, and then executes the keystroke itself. If the VAR menu contains more than six labels, use (NXT) and (↰)(PREV) to change menu pages.

Example: Create a variable named *OPTION* containing 6.011991 and display the VAR menu.

6.011991 (ENTER)
(') OPTION (STO)
(VAR)

Recall the value of the variable.

(↱) OPTIO

Enter the name of the variable.

(') OPTIO (ENTER)

Display a catalog of the variables.

(⬅)(REVIEW)

```
OPTION: 6.011991
ADD2: « 2 + »
ALG: 'A+B'
B: 5
A: 2
VCT1: [ 1 2 3 ]
OPTIO ADD2 ALG  B  A  VCT1
```

Press (ATTN) to return to the stack display.

To reorder the VAR menu:

1. Create a list containing variable names in the order you want them to appear in the VAR menu. The list doesn't have to include all the names—omitted names are positioned after included names.
2. Press (⬅)(MEMORY) ORDER (the ORDER command).

One way to create the list (step 1) is to execute the VARS command ((⬅)(MEMORY) VARS). VARS returns a list containing all the variables in the current directory. You can then edit the list.

Purging Variables

To purge one variable:

1. Enter the variable name.
2. Press (⬅)(PURGE) (the PURGE command).

Example: Purge the variable *OPTION* created in the previous example. (You can type the name or use the variable's menu key as a typing aid.)

(') OPTION (⬅)(PURGE)

or

(') OPTIO (⬅)(PURGE)

```
ADD2 ALG  B  A  VCT1 DIFF
```

6

To purge more than one variable:

1. Create a list (with { } delimiters) containing the names of the variables to be purged. (Press [ENTER] to put it in level 1.)
2. Press [⬅][PURGE] (the PURGE command).

To get a list of all variables in the current directory, press [⬅][MEMORY] VARS (the VARS command). You can edit this list, then execute PURGE.

Example: Suppose you had created the variables *A*, *B*, *C*, *D*, and *E*—and now you want to purge *A*, *B*, and *C*. First, create the list { A B C } containing the names to purge—press [⬅][{ }] A B C [ENTER]. (Notice that [{ }] places the command line in Program-entry mode.) Then press [⬅][PURGE] to delete those variables.

To purge all variables:

1. Press [➡][PURGE], a typing aid that displays CLVAR in the command line.
2. Press [ENTER] to execute the command.

The CLVAR (clear variables) command purges all the variables in the current directory. Since this command can erase a great deal of data, there's no immediate-execution key for it.

Recovering from Errors

To undo a STO or PURGE command for one variable:

1. Press [➡][LAST ARG] *before executing any other operations.*
2. Press [STO].

If you accidentally overwrite or purge a variable by pressing [STO] or [⬅][PURGE], then [➡][LAST ARG] returns the contents of the variable *prior* to the error and the variable name to the stack. Then, you can press [STO] to restore the variable.

6

Doing Variable Arithmetic

The variable arithmetic commands perform arithmetic operations on a variable's contents without retrieving the contents to the stack. The commands are located in the MEMORY Arithmetic menu ((→)(MEMORY)).

Variable Arithmetic Commands

Key	Programmable Command	Description
(→)(MEMORY):		
STO+ STO- STO* STO/	STO+ STO− STO* STO/	Adds, subtracts, multiplies, or divides two objects, where one is taken from the stack and the other is the contents of a variable specified by a name on the stack. The new object is the level 2 object plus, minus, times, or divided by the level 1 object, and it's stored in the specified variable.
SINV SNEG SCON	SINV SNEG SCONJ	Computes the inverse, negative, or complex conjugate of the contents of the variable named on the stack. The result replaces the original contents of the variable.

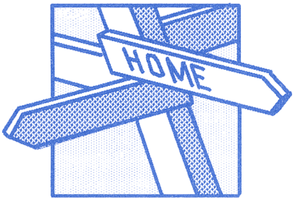
Variable Arithmetic Examples

Command	Previous Contents of ABC	Stack Contents	Final Contents of ABC
STO+	10	2: 'ABC' 1: 6	16
STO−	10	2: 3 1: 'ABC'	−7
STO/	20	2: 'ABC' 1: 2	10
	20	2: 5 1: 'ABC'	0.25

Two additional commands, INCR and DECR, are used primarily in programming. They're covered under "Using Loop Counters" on page 27-13.

7

Directories



Directories let you organize variables into meaningful groupings. They also let you "bury" information that you use infrequently and protect data that you don't want programs (or people) to alter accidentally.

This chapter covers these topics:

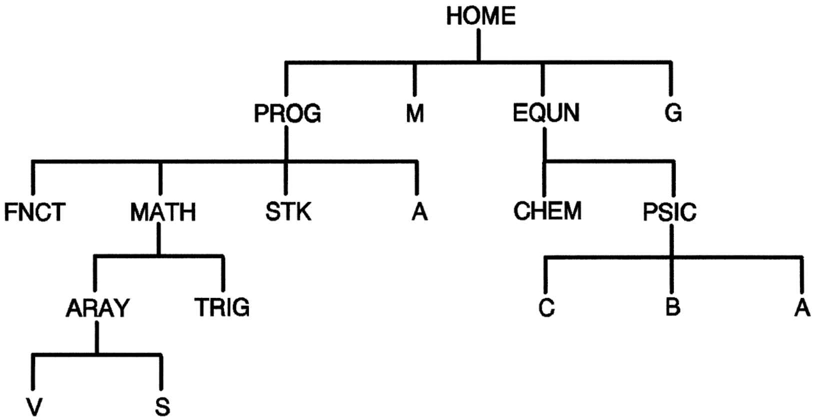
- Understanding what directories are.
- Creating subdirectories.
- Creating and accessing variables in directories.
- Changing, purging, and manipulating directories.

Learning about Directories

The HP 48 lets you organize your variables in a hierarchical structure—so that you can work with *collections* of variables, rather than *all* variables at once. This helps keep your VAR menu from getting too cluttered—and helps separate variables that aren't related.

A *directory* is an object containing a collection of variable names and corresponding stored objects. The *current directory* is the one directory that's active—the one whose variable names appear in the VAR menu.

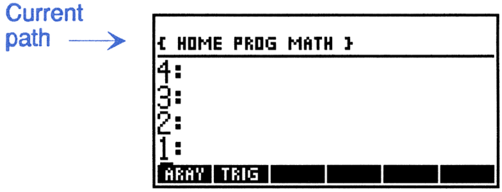
Consider the following directory structure.



The *HOME* directory is always the top-level directory. In this example, it contains four variables—two variables (*PROG* and *EQUN*) are names of subdirectories. Moving downward, *MATH* is a subdirectory of *PROG*, and *ARAY* is a subdirectory of *MATH*. (You can also say that *MATH* is the parent of *ARAY*, *PROG* is the parent of *MATH*, and *HOME* is the parent of *PROG*.)

The sequence of directories—starting from *HOME*—that leads to a directory is the *path* of that directory. The path of *EQUN* is { HOME EQUN }. The path of *ARAY* is { HOME PROG MATH ARAY }.

The path of the current directory is the *current path*—which is shown in the status area of the display.



7

A directory is normally stored in a variable—and when its name appears in the VAR menu, its menu label has a bar over the top-left corner to show that it's a directory.

The *HOME* directory is the only directory that exists when the calculator is turned on for the first time. You create other directories as needed.

To put the path of the current directory on the stack:

- Press (←)(MEMORY) PATH (the PATH command).

The PATH command returns the path of the current directory in the form of a list of directory names. For example, if *ARAY* were the current directory and you executed PATH, the list { HOME PROG MATH ARAY } would be returned to level 1. (You can evaluate a directory path with EVAL to switch to the directory specified by the path.)

Creating Subdirectories

To create a subdirectory in the current directory:

1. Enter a name for the subdirectory. (Press (') and type the name.)
2. Press (←)(MEMORY) CRDIR (the CRDIR command).

The new name is added to the VAR menu. A bar over the top-left corner of the menu label indicates that it's a directory.

When you create a variable, it's added to the *current directory*. If the variable name already exists in that directory, the new variable overwrites the previous contents.

Example: Create two subdirectories in the *HOME* directory. Name them *EQUN* and *PROG*.

If necessary, press (→)(HOME) to make *HOME* the current directory. (The status area should display { HOME }.) Then, display the VAR menu. (Your VAR menu may be different.)

(VAR) | ADD2 | ALG | B | A | VCT1 | DIFF |

Create the subdirectories.

[']EQUN [⬅][MEMORY] CRDIR
[']PROG CRDIR [VAR]

PROG EQUN ADD2 ALG B A

The names of the new directories have been added to the VAR menu. A bar over the left side of each label indicates that they're directories. Now, switch to the *EQUN* directory. Its VAR menu is initially empty.

EQUN

{ HOME EQUN }

Store 'Y=SIN(X)' into a variable named *WAVE*. Its label is placed in the VAR menu.

['] Y [⬅][=] [SIN] X [ENTER]
['] WAVE [STO]

WAVE

Accessing Variables in Directories

When you evaluate a name, the HP 48 searches the current directory for that name. If the name isn't there, the HP 48 searches the parent directory, continuing upwards, if necessary, all the way to the *HOME* directory. This provides several useful features (examples are taken from the diagram on page 7-2):

- A variable in the *HOME* directory can be accessed from any other directory as long as there is no other variable with the same name along the current path. For example, variables *M* and *G* can be accessed from anywhere. However, if the *PROG* directory contained another variable *M*, that variable would be used when *PROG*, *MATH*, or *ARAY* was the current directory.
- Variables beneath the current directory can't be accessed. For example, when *EQUN* is the current directory, you can't access variables in the *PSIC* directory.
- You can use duplicate variable names. For example, the two variables *A* are unrelated—one can be accessed from *MATH* and *ARAY*, the other from *PSIC*.

Changing Directories

To switch to any directory:

- Enter a list (with { } delimiters) starting with *HOME* and containing the path to the directory, then press [EVAL].

To switch to a subdirectory:

- Press the menu key for that directory in the VAR menu.

  or
- Enter the unquoted directory name and press [ENTER].

  or
- Enter a list containing the path to a lewer-level directory you want and press [EVAL].

To switch to a higher directory:

- To switch to the next higher directory, press [⬅][UP] (the UP command).
- To switch to the *HOME* directory, press [➡][HOME] (the HOME command).
- To switch to any higher directory along the current path, enter the unquoted directory name and press [ENTER].

Purging Variables and Directories

You use the PURGE command to delete selected variables in the current directory. See "Purging Variables" on page 6-8 for details.

But if you want to delete a directory, you usually must delete *all* variables in that directory first.

To purge all variables in a directory:

- Press [➡][PURGE]—a typing aid that displays CLVAR in the command line—then press [ENTER] to execute the command.

  or
- Press [⬅][MEMORY] VARS , then press [⬅][PURGE].

The CLVAR command purges all variables in the current directory. The VARS command returns a list containing all the variables in the current directory. Empty subdirectories are successfully purged by these steps. However, if the current directory contains a subdirectory that's not empty (it contains variables), you get an error, leaving that subdirectory as the first entry in the VAR menu.

To purge an empty directory:

1. Switch to its parent directory.
2. Enter the directory's name. (Press ['], then press the directory's menu key or type its name.)
3. Press [←] [PURGE].

Or, after a directory is empty, you can purge it with other variables using the PURGE or CLVAR command.

To purge a directory that's not empty:

1. *Make sure you know what you're deleting before you do this.*
2. Switch to its parent directory.
3. Enter the directory's name. (Press ['], then press the directory's menu key or type its name.)
4. Press [←] [MEMORY] [NXT] [NXT] PGDIR (the PGDIR command).

Using Directory Objects on the Stack

A subdirectory is a variable containing a *directory object*. Creating a subdirectory with CRDIR (create directory) is analogous to creating other variables with STO, except that you are specifically creating a variable containing an empty directory object. For example, ['] EQUN CRDIR creates a directory *EQUN* by storing an empty directory object into a variable named *EQUN*.

To recall a directory object to the stack:

- Press [→] followed by the directory's menu key in the VAR menu.

  or
- Enter the directory's name (press ['], then press the directory's menu key or type its name), then press [→] [RCL].

Directory objects are displayed as:

```
DIR
name1 object1
name2 object2
:
END
```

where $name_n$ is the name of a variable in the directory, and $object_n$ is the contents of that variable. The words DIR and END are the delimiters of the directory object. (You can also create or edit a directory of this form in the command line.)

Because subdirectories are variables containing a particular type of object, you can manipulate them like other variables. For example, you can recall them to the stack and then store them in another directory. This provides a way to copy or move subdirectories.

HOME is a special directory that is *not* a variable. Therefore, you can't manipulate it as an object the way you can manipulate other directories.

Example: Change the directory name *EQUN* to *BIO*.

Switch to the *HOME* directory, then recall the *EQUN* directory to the stack.

[↱] [HOME]
['] EQUN [↱] [RCL]

```
1: DIR
   WAVE 'Y=SIN(X)'
   END
PROG | EQUN | ADD2 | ALG | B | A
```

Store it using the new name.

['] BIO [STO]

```
BIO | PROG | EQUN | ADD2 | ALG | B
```

Purge the old directory, then check the VAR menu.

['] EQUN
[↰] [MEMORY] [NXT] [NXT] PGDIR
[VAR]

```
BIO | PROG | ADD2 | ALG | B | A
```

Note

If you recall a directory to the stack and then change the directory contents, the copy on the stack will change as well.

8

More about Algebraic Objects

Algebraic objects (*algebraics* for short) are the vehicle for symbolic mathematics in the HP 48. This chapter addresses topics that will help you understand better the behavior of algebraics:

- Evaluation of algebraics.
- Rules of algebraic precedence.
- Expressions and equations.

Entering Algebraics

You can enter equations and expressions in the command line—or you can use the EquationWriter application. (The EquationWriter application is described in chapter 16.) Algebraic objects are delimited by two ' marks.

To enter an algebraic in the command line:

1. Press ['].
2. Key in the numbers, variables, operators, and parentheses in the expression or equation in left-to-right order. Press [▶] to skip past right parentheses.
3. Press [ENTER].

For algebraic syntax, arguments usually appear in their common locations: before and after functions like + and /, or inside parentheses after functions like SIN and MAX.

Evaluation of Algebraics

Evaluation moves an algebraic towards its numeric value.

To evaluate an algebraic in level 1 (or in the command line):

- Press (EVAL) (the EVAL command).

The Evaluation Process

To understand what to expect when you evaluate an algebraic, recognize that an algebraic is equivalent to a *program* (introduced in chapter 1). A program is simply a series of objects enclosed by « » delimiters. Evaluating a program means: "Put each object in the program on the stack, and, if the object is a command or unquoted name, evaluate it." The same procedure is carried out when an algebraic is evaluated. Names in algebraics normally aren't quoted (if necessary, you can use the QUOTE function to prevent evaluation).

Suppose variable X contains the value 3, and Y has value 4. When you execute `'X+Y' EVAL`, `7` is returned to the stack. Here's how:

1. The name X is evaluated, returning `3` to level 1 of the stack.
2. Y is evaluated, returning `4` to level 1 and pushing `3` to level 2.
3. + is evaluated, taking the arguments `3` and `4` from the stack and returning `7`.

Now suppose variable X contains the value 3 and variable T is *formal*—meaning it has no value stored in it, it doesn't exist. When you execute `'X-T' EVAL`, `'3-T'` is returned to the stack. Here's how:

1. X is evaluated, returning `3` to level 1.
2. T is evaluated. Because T has no value associated with it, it just returns `'T'` to level 1, pushing `3` to level 2.
3. This time − takes arguments `3` and `'T'` from the stack. Because `'T'` is an algebraic, − returns an algebraic `'3-T'` to level 1.

Stepwise Evaluation

Evaluation is a stepwise process—EVAL causes only one level of substitution.

Example: Evaluate '√A*B', where *A* contains 'B+5', *B* contains 'X/2', and *X* contains 3.

Create the three variables.

['] B [+] 5 [ENTER] ['] A [STO]
['] X [÷] 2 [ENTER] ['] B [STO]
3 ['] X [STO]
[VAR]

Evaluate '√A*B'. Each occurrence of *A* evaluates to 'B+5' and each occurrence of *B* evaluates to 'X/2'.

['] [√x] A [×] B
[EVAL]

Evaluate the algebraic again. Once again, each occurrence of *B* evaluates to 'X/2'. Furthermore, each occurrence of *X* evaluates to 3.

[EVAL]

Evaluate again to complete the process.

[EVAL]

Symbolic and Numeric Results

In the previous example, the HP 48 was in *Symbolic Results* mode—repeated evaluation resulted in the progressive resolution of symbolic terms until a numeric result was obtained. This is the default state for the calculator. If you select *Numeric Results* mode, algebraics evaluate *directly to a number* in one step.

To switch to Numeric Results mode or to Symbolic Results mode:

- Press [←] [MODES] SYM .

SYM■ means Symbolic Results mode is active. SYM means Numeric Results mode is active.

Or, you can set or clear flag −3 to change the results mode. The results mode governs execution of *functions*—algebraics are affected indirectly.

Example: Evaluate the algebraic from the previous example in Numeric Results mode. Then restore Symbolic Results mode.

[←] [MODES] SYM■
['] [√x] A [×] B
[EVAL] SYM

Note that in Numeric Results mode, evaluation of an algebraic that contains a *formal* variable (one in which no object is stored, one that doesn't exist) generates an error because that variable prevents obtaining a numeric result. For example, in Numeric Results mode, evaluation of 'X+T' where X has value 3 and T is formal leaves 3 in level 2 and 'T' in level 1, and displays the message:

```
+ Error:
Undefined Name
```

To evaluate an algebraic directly to a numeric result:

- Press [→] [→NUM] (the →NUM command).

The →NUM command evaluates an algebraic in level 1 (or in the command line) directly to a numeric result regardless of the results mode:

1. Switches to Numeric Results mode (if Symbolic Results mode is active).
2. Executes EVAL.
3. Turns Symbolic Results mode back on (if it was on before execution of →NUM).

Automatic Simplification

When you evaluate certain functions, they replace certain symbolic arguments or combinations of arguments with simpler forms. For example, when you evaluate '1*X', the * function detects that one of its arguments is 1, so the expression is replaced by 'X'. The following table shows several examples of automatic simplification.

Original Expression	Simplified Expression
'SQ(√X)'	'X'
'X-X'	0
'X*INV(Y)'	'X/Y'
'X^(-1)'	'INV(X)'
'COS(-X)'	'COS(X)'
'ABS(-X)'	'ABS(X)'
'EXP(LN(X))'	'X'
'CONJ(RE(X))'	'RE(X)'

Rules of Algebraic Precedence

The precedence of operators in an algebraic determines the order of evaluation of terms. Operations with higher precedence are performed first. Algebraics are evaluated from left to right for operators with the same precedence. The following list gives HP 48 functions in order of precedence, from highest (1) to lowest (11):

1. Expressions within parentheses. Expressions within nested parentheses are evaluated from inner to outer.
2. Functions like SIN or LOG that require arguments in parentheses.
3. ! (factorial).
4. Power (^) and square root (√).
5. Negation (−), multiplication (*), and division (/).
6. Addition (+) and subtraction (−).
7. Comparison operators (== ≠ < > ≤ ≥).

8. Logical operators AND and NOT.
9. Logical operators OR and XOR.
10. The left argument for | (where).
11. Equals (=).

Example:

`'A^3+B'`	Cubes A, then adds B to that quantity, since ^ has a higher precedence than +.
`'A^(3+B)'`	Raises A to the power $3+B$, since an expression within parentheses has a higher precedence than ^.

Expressions and Equations

An *expression* is an algebraic that does not contain an = function. An *equation* is an algebraic that does contain an = function. For example, `'SIN(X)-ATAN(2*X)+6*X'` is an expression, and `'SIN(X)=ATAN(2*X)+6*X'` is an equation.

When you use an equation as the argument of a function, the function is applied to both sides, and the result is also an equation. For example, `'X=Y'` `SIN` returns `'SIN(X)=SIN(Y)'`.

In the HP 48, the = sign generally means equating two expressions. The DEFINE command ([←][DEF]) interprets = differently—it *stores* the expression on the right side of the = sign in the name on the left side.

Related Topics

This chapter does not cover all aspects of algebraic objects—they're used in many different ways in the HP 48. You can find related topics in the following sections of the manual:

- "Using Symbolic Constants" on page 9-15.
- "Using Symbolic Arguments with Common Math Functions" on page 9-18.
- "Using Complex Numbers in Algebraics" on page 11-7.
- "Using Unit Objects in Algebraics" on page 13-7.

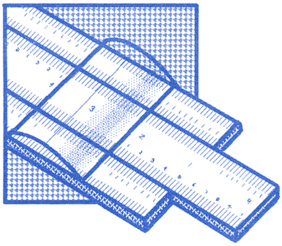
In addition, in chapters 17, 18, 19, 22, 23, and 25 through 31, you'll see how algebraics are used extensively in the HP Solve application, plotting, algebra, calculus, and programming.

Part 2

Hand Tools

9

Common Math Functions



The MTH menu ([MTH]) is a menu of four specific mathematical menus. Many of the common math functions described in this chapter either are found on the keyboard or are located in the PARTS, PROB, HYP, and VECTR menus, which are submenus of the MTH menu. Press [MTH] to see menu labels for these submenus.

Algebraic Syntax and Stack Syntax

As described under "Built-In Functions and Commands" on page 4-10, functions are a subset of commands. The difference between functions and other commands is that functions can be included in algebraics. This means that functions can take their arguments from the stack like other commands, or they can be executed in algebraic syntax as part of a symbolic expression.

Example: Algebraic Syntax. Calculate the sine of 30° using a symbolic expression.

Make sure Degrees mode is set.

[←] [MODES] [NXT] [NXT] DEG

```
DEG■ RAD GRAD XYZ■ R∠Z R∠∠
```

Activate Algebraic-entry mode and enter the expression, supplying the argument in algebraic syntax.

['] [SIN] 30 [ENTER]

```
1:                'SIN(30)'
DEG■ RAD GRAD XYZ■ R∠Z R∠∠
```

Evaluate the expression.

(EVAL)

```
1:                    .5
DEG■ RAD GRAD XYZ■ R∠Z R∠∠
```

Example: Stack Syntax. Calculate the sine of 30° using an argument from the stack.

Enter the argument on the stack.

30 (ENTER)

```
2:                    .5
1:                    30
DEG■ RAD GRAD XYZ■ R∠Z R∠∠
```

Calculate the sine of the argument.

(SIN)

```
2:                    .5
1:                    .5
DEG■ RAD GRAD XYZ■ R∠Z R∠∠
```

Keep in mind as you work through the rest of this chapter that you can execute these functions both ways. The rest of this chapter describes:

- The various sets of functions for manipulating real numbers. You can also use many of these functions with other object types.
- The HP 48 built-in symbolic constants—π, e, i, MAXR (maximum real number), and MINR (minimum real number).

Arithmetic and General Math Functions

Many arithmetic and general math functions are found on the keyboard.

Arithmetic and General Math Functions

Key	Programmable Command	Description
[1/x]	INV	Inverse.
[√x]	√	Square root.
[←] [x^2]	SQ	Square.
[+/-]	NEG	Change sign. Changes the sign of the number in the command line. When no command line is present, [+/-] executes a NEG command (changes the sign of the argument in level 1).
[+]	+	Level 2 + level 1.
[-]	−	Level 2 − level 1.
[×]	*	Level 2 × level 1.
[÷]	/	Level 2 ÷ level 1.
[y^x]	^	Level 2 raised to the level 1 power. The algebraic syntax for the ^ command is 'y^x'.
[$\sqrt[x]{y}$]	XROOT	The xth (in level 1) root of a real value in level 2. The algebraic syntax for the XROOT command is 'XROOT(x,y)'.

Example: Calculate $2.7^{1.1\times1.6}$. First, enter 2.7.

2.7 [ENTER]

Next, calculate 1.1 × 1.6.

1.1 [ENTER]
1.6 [×]

Now, do the exponentiation.

y^x

```
1:            5.74381210967
DEG ■  RAD  GRAD  XYZ ■  R∠Z  R∠∠
```

Example: Calculate $\sqrt[3]{28}$.

28 [ENTER] 3 [→] [$\sqrt[x]{y}$]

```
1:            3.03658897188
DEG ■  RAD  GRAD  XYZ ■  R∠Z  R∠∠
```


Example: Enter the complex number (2,4) and negate it.

[←] [( )] 2 [SPC] 4 [ENTER]
[+/–]

```
1:                  (-2,-4)
DEG ■  RAD  GRAD  XYZ ■  R∠Z  R∠∠
```

Compare the previous results to what happens when you press [+/–] immediately after keying in the 4.

[←] [( )] 2 [SPC] 4 [+/–] [ENTER]

```
2:                  (-2,-4)
1:                   (2,-4)
DEG ■  RAD  GRAD  XYZ ■  R∠Z  R∠∠
```

Example: Use a function name (unevaluated) in an algebraic expression.

['] [$\sqrt{x}$] 5 [ENTER]

```
1:                     '√5'
DEG ■  RAD  GRAD  XYZ ■  R∠Z  R∠∠
```

Evaluate the expression.

[EVAL]

```
1:             2.2360679775
DEG ■  RAD  GRAD  XYZ ■  R∠Z  R∠∠
```

Fraction Conversion Functions

Two functions enable you to find a "best-guess" fractional approximation to a real number. The fraction is returned as an algebraic expression involving the division of two integers.

Fraction Conversion Functions

Key	Programmable Command	Description
[↰] [→Q]	→Q	Quotient approximation. Returns "best-guess" fraction for a real number, reflecting the accuracy implied by the display mode.
[↰] [ALGEBRA] (page 2):		
→Qπ	→Qπ	Quotient-π approximation. Returns "best-guess" fraction for a real number, possibly including π and reflecting the accuracy implied by the display mode.

The accuracy of the fractional approximation is dependent on the display mode. If the display mode is Standard ([↰] [MODES] STD), the approximation is accurate to 11 significant digits. If the display mode is n Fix, the approximation is accurate to n significant digits.

→Qπ computes both the fractional equivalent of the original number *and* the fractional equivalent of the original number divided by π, and then compares the denominators. It returns the fraction with the smallest denominator—this fraction might be the same fraction returned by →Q, or it might be a different fraction multiplied by π.

Example: Convert 7.896 to a pure fraction using →Q.

7.896 [↰] [→Q]

```
1:                '987/125'
DEG ■ | RAD | GRAD | XYZ ■ | R∠Z | R∠∠
```

Exponential, Logarithmic, and Hyperbolic Functions

Exponential and Logarithmic Functions

Key	Programmable Command	Description
⬅ (10^x)	ALOG	Common (base 10) antilogarithm.
➡ (LOG)	LOG	Base 10 logarithm.
⬅ (e^x)	EXP	Natural (base e) antilogarithm.
➡ (LN)	LN	Natural (base e) logarithm.

Hyperbolic Functions

Key	Programmable Command	Description
(MTH) HYP :		
SINH	SINH	Hyperbolic sine: $(e^x - e^{-x})/2$.
ASINH	ASINH	Inverse hyperbolic sine: $\sinh^{-1} x$.
COSH	COSH	Hyperbolic cosine: $(e^x + e^{-x})/2$.
ACOSH	ACOSH	Inverse hyperbolic cosine: $\cosh^{-1} x$.
TANH	TANH	Hyperbolic tangent: $\sinh x / \cosh x$.
ATAN	ATANH	Inverse hyperbolic tangent: $\sinh^{-1}(x/\sqrt{1-x^2})$.
EXPM	EXPM	$e^x - 1$. Argument x is in level 1. (EXPM is more accurate than EXP when the argument to e^x is close to 0.)
LNP1	LNP1	$\ln(x+1)$. Argument x is in level 1. (LNP1, ln *plus* 1, is more accurate than LN when the argument to ln is close to 1.)

Example: Calculate the hyperbolic sine of 5.

5 (MTH) HYP SINH

```
1:            74.2032105778
SINH ASINH COSH ACOSH TANH ATAN
```

Percent Functions

Percent Functions

Key	Programmable Command	Description
MTH PARTS (page 2):		
%	%	A percent of B, or B percent of A (A is in level 2, B is in level 1): $(A \times B)/100$.
%CH	%CH	The percent change from A to B, as a percentage of A (A is in level 2, B is in level 1): $((B - A)/A) \times 100$.
%T	%T	The percent of total (the total, A, is in level 2 and the value, B, is in level 1): $(B/A) \times 100$.

9

Example: Calculate 12.5% of 650.

650 ENTER 12.5
MTH PARTS NXT %

```
1:                    81.25
MIN MAX MOD % %CH %T
```

Example: Calculate the percent change between 8 and 8.5.

8 ENTER 8.5 %CH

```
1:                     6.25
MIN MAX MOD % %CH %T
```

Example: If 35 out of 500 units fail a test, what percentage failed?

500 ENTER
35 %T

```
1:                        7
MIN MAX MOD % %CH %T
```

Trigonometric Functions, Angle Mode, and π

Selecting the Angle Mode

The angle mode determines how the calculator interprets angle
9 arguments and how it returns angle results.

Angle Modes

Mode	Definition	Annunciator
Degrees	$^1/_{360}$ of a circle.	(none)
Radians	$^1/_{2\pi}$ of a circle.	RAD
Grads	$^1/_{400}$ of a circle.	GRAD

To change the angle mode from the keyboard:

- Press [↰] [RAD] to switch between Radians mode and Degrees mode. (If Grads mode had been previously selected in the MODES menu, this switches between Radians mode and Grads mode instead.)

  or

- Press [↰] [MODES] [NXT] [NXT], then DEG, RAD, or GRAD. The ■ in the menu label indicates the active mode.

Trigonometric Functions

For trigonometric functions, the angle arguments and results are interpreted as degrees, radians, or grads, depending on the current angle mode.

Trigonometric Functions

Key	Programmable Command	Description
[SIN]	SIN	Sine.
[↰] [ASIN]	ASIN	Arc sine.
[COS]	COS	Cosine.
[↰] [ACOS]	ACOS	Arc cosine.
[TAN]	TAN	Tangent.
[↰] [ATAN]	ATAN	Arc tangent.

Example: Set Radians mode, then calculate the sine of 1.1 radians.

[↰] [MODES] [NXT] [NXT] RAD
1.1 [SIN]

The Constant π

The number π cannot be represented exactly in a finite number of decimal places. The calculator provides a 12-digit approximation (3.14159265359) to π. The HP 48 also provides a symbolic constant that represents π exactly.

In *Radians* mode, the SIN, COS, and TAN functions recognize the argument π and return exact results. SIN and COS also recognize $^{\pi}/_{2}$.

To enter the symbolic constant 'π' in level 1:

- Press [↰] [π].

If Numeric Results mode is active (the MODES menu shows SYM, flag −3 is set), you get the numeric approximation instead.

To replace 'π' with its 12-digit value:

- Press [↱] [→NUM] (the →NUM *to-number* command).

See "Using Symbolic Constants" on page 9-15 for more information about π and other symbolic constants.

Example: Calculate cos ($^{\pi}/_{2}$) and cos ($^{\pi}/_{4}$). (This example assumes the calculator is in Symbolic Results mode—the SYM■ label on page 1 of the MODES menu has a ■ in it.)

If necessary, switch to Radians mode. Then, put `'π'` in level 1 and divide it by 2.

9

⬅ MODES NXT NXT RAD
⬅ π 2 ÷

```
1:                'π/2'
DEG  RAD■  GRAD  XYZ■  R∠Z  R∠∠
```

Calculate the cosine.

COS

```
1:                    0
DEG  RAD■  GRAD  XYZ■  R∠Z  R∠∠
```

Now, enter $^{\pi}/_{4}$.

⬅ π 4 ÷

```
2:                    0
1:                'π/4'
DEG  RAD■  GRAD  XYZ■  R∠Z  R∠∠
```

Now, calculate cos ($^{\pi}/_{4}$).

COS

```
2:                    0
1:           'COS(π/4)'
DEG  RAD■  GRAD  XYZ■  R∠Z  R∠∠
```

The HP 48 retains the symbolic constant π and returns a symbolic expression. Use →NUM to calculate a numeric result.

➡ →NUM

```
2:                    0
1:        .707106781186
DEG  RAD■  GRAD  XYZ■  R∠Z  R∠∠
```

Press DEG to switch back to Degrees mode.

Angle Conversion Functions

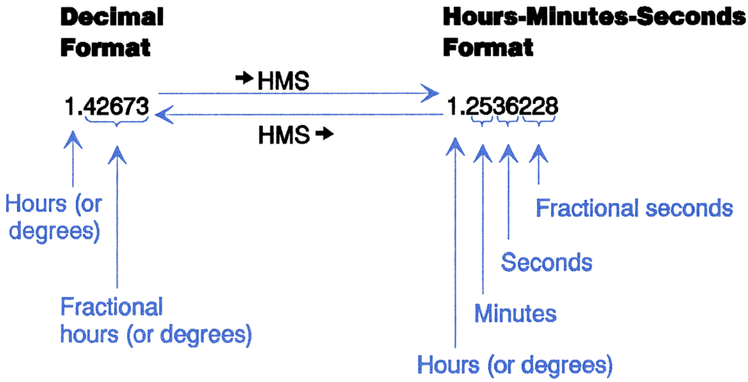
Two commands in the MTH VECTR menu convert values between decimal degrees and radians. Four other commands in the TIME menu let you do degrees-minutes-seconds calculations using hours-minutes-seconds (HMS) format. See "Making Time and Angle Calculations" on page 24-18.

In Degrees mode, angle arguments and results use *decimal degrees.*

Angle Conversion Functions

Key	Programmable Command	Description
MTH VECTR (page 2):		
D→R	D→R	Degrees to radians. Converts a number from a decimal degree value to its radian equivalent.
R→D	R→D	Radians to degrees. Converts a number from a radian value to its decimal degree equivalent.
← TIME (page 3):		
→HMS	→HMS	Decimal to HMS. Converts a number from decimal degrees to HMS format.
HMS→	HMS→	HMS to decimal. Converts a number from HMS format to decimal degrees.
HMS+	HMS+	Adds two numbers in HMS format.
HMS−	HMS−	Subtracts two numbers in HMS format.

The following illustrates the conversion to and from HMS format:



Example: Convert 1.79π radians to degrees. First, enter 1.79π.

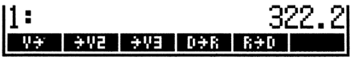
1.79 (ENTER)
(⬑)(π)(×)

Use the R→D function. (The function acts independently of the current angle mode.)

(MTH) VECTR (NXT) R→D

Use →NUM to obtain a numeric result.

(⬏)(→NUM)



Example: Add 25.2589 degrees to 34 degrees, 55 minutes, 31.22 seconds.

Convert 25.2589 degrees to HMS format.

25.2589 (⬑)(TIME) (NXT) (NXT)
→HMS

Add 34 degrees, 55 minutes, and 31.22 seconds to the result.

34.553122 HMS+

Factorial, Probability, and Random Numbers

Factorial, probability, and random number commands are found in the MTH PROB menu ((MTH) PROB).

Probability Commands

Key	Programmable Command	Description
(MTH) PROB :		
COMB	COMB	Number of combinations of n (in level 2) items taken m (in level 1) at a time.
PERM	PERM	Number of permutations of n (in level 2) items taken m (in level 1) at a time.
!	!	Factorial of a positive integer. For non-integers, ! returns $\Gamma(x + 1)$.
RAND	RAND	Returns the next real number n $(0 \leq n < 1)$ in a pseudo-random number sequence. Each random number becomes the seed for the next random number.
RDZ	RDZ	Takes a real number from level 1 as a seed for the next random number (from RAND). 0 in level 1 creates a seed based on the clock time. A sequence of random numbers can be repeated by starting with the same nonzero seed.

Example: Calculate the number of combinations and permutations of 10 objects taken 4 at a time.

(MTH) PROB
10 (ENTER) 4 COMB
(⮫) (LAST ARG) PERM

```
2:                210
1:               5040
COMB PERM  !  RAND RDZ
```

Other Real-Number Functions

The functions in the following table are found in the MTH PARTS menu ((MTH) PARTS).

9

Command/Description	Example	
	Input	Output
ABS Absolute value.	1: -12	1: 12
CEIL Smallest integer greater than or equal to the argument.	1: -3.5 1: 3.5	1: -3 1: 4
FLOOR Greatest integer less than or equal to the argument.	1: 6.9 1: -6.9	1: 6 1: -7
FP Fractional part of the argument.	1: 5.234 1: -5.234	1: .234 1: -.234
IP Integer part of the argument.	1: -5.234 1: 5.234	1: -5 1: 5
MANT Mantissa of the argument.	1: 1.23E12	1: 1.23
MAX Maximum; the greater of two arguments.	2: 5 1: -6	1: 5
MIN Minimum; the lesser of two arguments.	2: 5 1: -6	1: -6
MOD Modulo; remainder of $^A/_B$. A MOD B = $A - B$ FLOOR $(^A/_B)$.	2: 6 1: 4	1: 2

Command/Description	Example	
	Input	Output

RND Rounds number according to argument: $0 \leq n \leq 11$ rounds to n FIX, $-11 \leq n \leq -1$ rounds to n significant digits, and $n = 12$ rounds to the current display format.	`2: 1.2345678`
	`1: 5`
	
	`2: 1.2345678`
	`1: -5`
		`1: 1.23457`
		
		`1: 1.2346`

SIGN Returns +1 for positive arguments, −1 for negative arguments, and 0 for arguments of 0.	`1: -2.7`	`1: -1`

TRNC Truncates number according to argument: $0 \leq n \leq 11$ truncates to n FIX, $-11 \leq n \leq -1$ truncates to n significant digits, and $n = 12$ truncates to the current display format.	`2: 1.2345678`
	`1: 5`
	
	`2: 1.2345678`
	`1: -5`
		`1: 1.23456`
		
		`1: 1.2345`

XPON Exponent of the argument.	`1: 1.23E45`	`1: 45`

9

Using Symbolic Constants

The HP 48 has five built-in constants: π, e, i, MAXR (maximum real number), and MINR (minimum real number). Use lowercase letters for i and e. The examples under "The Constant π" on page 9-9 show the use of π. The constant i is covered under "Using Complex Numbers in Algebraics" on page 11-7.

Example: The following keystrokes calculate $e^{2.5}$ two ways—using (e^x) and using e.

First, use the keyboard function.

2.5 (⬅)(e^x)

Second, enter a symbolic expression using e. (The keystrokes for the letter "e" are (α) (⬅) E.)

(') e (y^x) 2.5 (ENTER)

Execute →NUM to completely evaluate the expression to a number.

(➡)(→NUM)

Using Values for Symbolic Constants

You can use the constants in their symbolic form or as their machine-approximated values. When the MODES menu shows SYM■, which is its default state, functions operating on symbolic constants return symbolic results. This state is called *Symbolic Results mode*. When the menu shows SYM, *Numeric Results mode* is active—functions return numeric results.

To switch to Numeric Results mode or to Symbolic Results mode:

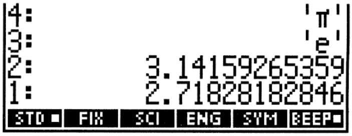
- Press (⬅)(MODES) SYM.

Example: Assuming Symbolic Results mode is currently active (the MODES menu shows SYM■), compare the effects of entering π and e in Symbolic Results mode and in Numeric Results mode.

Enter π and e in Symbolic Results mode. (The keystrokes for the letter "e" are (α) (⬅) E.)

(⬅)(π) e (ENTER)

Enter π and e in Numeric Results mode.



[←] [MODES] SYM■
[←] [π] e [ENTER]

Press SYM to restore Symbolic Results mode.

Using Flags to Interpret Symbolic Constants

System flags −2 (Symbolic Constants) and −3 (Numeric Results) control whether evaluating symbolic constants return symbolic or numeric results. The default setting for both flags is "clear."

To control the evaluation of symbolic constants:

- To leave a symbolic constant unchanged, clear flags −3 and −2 (their default states). Press [→] [→NUM] to replace the constant with its numeric value.
- To replace a constant with its numeric value, set flag −3.
- To replace a constant with its numeric value *except* when it's the argument of a function, clear flag −3 and set flag −2. Press [EVAL] *or* [→] [→NUM] to replace the function with the numeric result.

The →NUM command always returns a numeric result, regardless of the flag settings. (The MODES menu always shows the state of flag −3: SYM■ if clear, SYM if set.)

Example: To see the effect of the flag settings, clear both flags and enter π. (Note the *right* shift.)

2 [+/−] [→] [MODES] [NXT] CF
3 [+/−] CF
[←] [π]

Now, set flag −2 and press [←] [π]. It produces a numeric result.

2 [+/−] SF
[←] [π]

Now, enter the *expression* π. Because flag −3 is clear, the result is symbolic.

['] [←] [π] [ENTER]

9 Divide the symbolic π by 2.

2 [÷]

EVAL returns a numeric result with the current flag settings.

[EVAL]

Press 2 [+/−] CF to return to the default flag settings. (If you set flag −3, press 3 [+/−] CF .)

Using Symbolic Arguments with Common Math Functions

Functions that take real numbers as arguments also take symbolic arguments in the same way. For example, if ABS were executed with an argument of X instead of a number, the expression `'ABS(X)'` would be returned to the stack. Then, if the variable X contained a value, pressing [EVAL] would evaluate the expression for that value. (If flag −3 is set, a function taking a symbolic argument from the stack automatically evaluates to a number, if possible, when the function is executed.)

10

User-Defined Functions

The HP 48 lets you complement the built-in functions (such as SIN, LN, and MAX) with your own *user-defined functions*. A user-defined function behaves like a built-in function is several ways:

- It takes its arguments from the stack or in algebraic syntax.
- It takes symbolic arguments.
- It can be differentiated.

Creating a User-Defined Function

The DEFINE command lets you create a user-defined function directly from an equation. The equation must have the form '*name*(*arguments*)=*expression*'.

To create a user-defined function:

1. Enter an equation that specifies the function name and its arguments on the left side, and the expression that defines the calculation on the right side. On the left side, use commas to separate multiple arguments.
2. Press [⬅][DEF] (the DEFINE command).

Example: Use DEFINE to create *CMB*, a user-defined function that calculates the number of combinations C of n different items taken 1, 2, 3, … n at a time: $C = 2^n - 1$.

Enter the equation for *CMB*. (Use [←] to type lowercase letters.)

['] CMB [←][( )] n [▶]
[←][=] 2 [y^x] n [−] 1
[ENTER]

`1: 'CMB(n)=2^n-1'`
`TMEN RCLM STOF RCLF SF CF`

10 Execute DEFINE. Select the VAR menu and note that it now contains the user-defined function *CMB*.

[←][DEF]
[VAR]

`CMB BID PROG ADD2 ALG B`

Example: Create a user-defined function to calculate the surface area of a cylinder from its radius and height. Enter `'SCYL(r,h)=2*π*r*h+2*πr^2'` and press [←][DEF].

Executing a User-Defined Function

A user-defined function is executed just like a built-in function—it can take numeric or symbolic arguments, either from the stack or in algebraic syntax.

To execute a user-defined function:

- To use the stack, put the arguments on the stack in the same order they appear in the left side of the function definition (the last argument should be in stack level 1), then press the function key in the VAR menu (or type the function name and press [ENTER]).
- To use algebraic syntax, press ['], press the function key in the VAR menu (or type the function name), press [←][( )], enter the algebraic arguments in their proper order and separated by commas, then press [ENTER] (or press [EVAL] to evaluate the expression).

Example: Execute the user-defined function *CMB* from the earlier example to make the following calculations.

Calculate the total number of ways to combine one or more of four items ($n = 4$).

4 CMB

`1: 15`
`CMB BID PROG ADD2 ALG B`

For the same value of n, calculate the combinations in algebraic syntax.

['] CMB [⬅][( )] 4
[EVAL]

```
2:                15
1:                15
CMB  BIO  PROG  ADD2  ALG  B
```

Calculate *CMB(Z)* in algebraic syntax, where Z is a *formal variable.* (Purge Z to make sure is doesn't contain an object.)

10

['] Z [⬅][PURGE]
['] CMB [⬅][( )] Z
[EVAL]

```
1:            '2^Z-1'
CMB  BIO  PROG  ADD2  ALG  B
```

Differentiating a User-Defined Function

User-defined functions operate much like built-in functions—you can even differentiate user-defined functions. Differentiation is covered in detail under "Differentiating Expressions" on page 23-1.

Example: Calculate

$$\frac{d}{dx}\cot x$$

where $\cot x = \cos x / \sin x$. The HP 48 has no built-in cotangent function. However, you can create a user-defined function for cotangent. (This example assumes that variable X doesn't exist in the current directory—you can press ['] X [⬅][PURGE].)

First, select Radians mode. Then, enter the defining equation for the cotangent function.

[⬅][RAD] (if necessary)
['] COT [⬅][( )] X [▶] [⬅][=]
[COS] X [▶] [÷] [SIN] X [ENTER]

```
1: 'COT(X)=COS(X)/SIN(
   X)'
CMB  X  B  A  WAVE
```

Execute DEFINE to create the user-defined function *COT*. Next, enter the expression `'COT(X)'`, enter the variable name *X* to indicate the variable of differentiation, then perform the differentiation. (If variable *X* exists, you probably won't get this answer.)

[⬅] [DEF]
[VAR] ['] [COT] [⬅] [()] X [ENTER]
['] X [ENTER]
[➡] [∂]

```
1: '-(SIN(X)/SIN(X))-
   COS(X)*COS(X)/SIN(X
   )^2'
 COT | CMB | BIO | PROG | ADD2 | ALG
```

Press [⬅] [RAD] to return to Degrees mode.

You can use any variable as an argument to *COT*—that variable is automatically substituted into the original definition for *COT*.

Nesting User-Defined Functions

Just like built-in functions, user-defined functions can be included in the defining expression of a user-defined function.

Example: Write a user-defined function to calculate the ratio of surface area to volume of a box. The formula for this calculation is

$$\frac{A}{V} = \frac{2(hw + hl + wl)}{hwl}$$

where h, w, and l are the height, width, and length of the box.

First, create a user-defined function *BOXS* to calculate the surface area of the box. Use the EquationWriter application to key in the equation. (Use [⬅] to type lowercase letters.)

[⬅] [EQUATION]
BOXS [⬅] [()] h [SPC] w [SPC] l [▶]
[⬅] [=] 2 [×] [⬅] [()] h [×] w [+] h
[×] l [+] w [×] l

```
S(h,w,l)=2·(h·w+h·l+w·l)
 COT | CMB | BIO | PROG | ADD2 | ALG
```

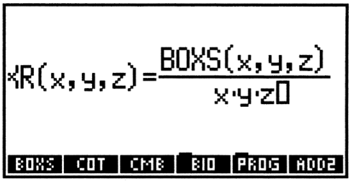
Enter the equation and create the user-defined function.

[ENTER]
[⬅] [DEF]

```
 BOXS | COT | CMB | BIO | PROG | ADD2
```

Now create a user-defined function *BOXR* to calculate the ratio of surface area to volume. Use the EquationWriter application to key in the equation.

[⬅][EQUATION]
BOXR [⬅][()] x [SPC] y [SPC] z [▶]
[⬅][=] [VAR] BOXS [⬅][()] x
[SPC] y [SPC] z [▶] [÷] x [×] y [×] z



Enter the equation and create the user-defined function.

[ENTER]
[⬅][DEF]

Use *BOXR* to calculate the ratio of surface area to volume for a box 9 inches high, 18 inches wide, and 21 inches long. Enter the height, width, and length, then execute *BOXR*.

9 [ENTER] 18 [ENTER] 21
[VAR] BOXR

Note that *BOXS* was defined using h, w, and l as variables, and that *BOXS* takes x, y, and z as arguments in the definition for *BOXR*. It makes no difference if the variables in the two definitions match—each set of variables is independent of the other.

The Structure of a User-Defined Function

A user-defined function is actually a *program*:

- It consists solely of a local variable structure whose defining procedure is a symbolic expression. The syntax is:

  « → $name_1$ $name_2$ ... $name_n$ '*expression*' »

- It takes an unlimited number of arguments (can use an unlimited number of local variables), but returns *one* result to the stack.

Local variables are described under "Using Local Variables" on page 25-13.

Example: Use VISIT to see the structure of user-defined function *CMB* from the example on page 10-1.

(VAR) (') CMB (↱)(VISIT)

```
« → n '2^n-1'
»
←SKIP SKIP→ ←DEL DEL→ INS ■ ↑STK
```


Press (ATTN) to return to the stack display.

You can see that the command sequence

```
'CMB(n)=2^n-1' DEFINE
```

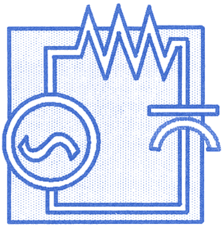
is equivalent to creating the program

```
« → n '2^n-1' »
```

and storing it in *CMB*.

11

Complex Numbers



Most functions that work with real numbers also work with complex numbers. So, the way you use complex numbers is similar to the way you use real numbers.

This chapter covers:

- Entering complex numbers.
- Interpreting and controlling the display format of complex numbers.
- Assembling and taking apart complex numbers.
- Calculating with complex numbers.
- Using complex numbers in algebraics.
- Determining when to use complex numbers and when to use vectors.

The examples in this chapter assume the calculator is set to Degrees mode. (Press [⬅][MODES] [NXT] [NXT] DEG to set Degrees mode.)

Displaying Complex Numbers

You can display complex numbers as either rectangular coordinates or polar coordinates—in *Rectangular mode* or in *Polar mode*.

To display rectangular coordinates for complex numbers:

- Press [➡][POLAR] until *no* coordinate annunciator is on.

  or
- Press [⬅][MODES] [NXT] [NXT] XYZ .

To display polar coordinates for complex numbers:

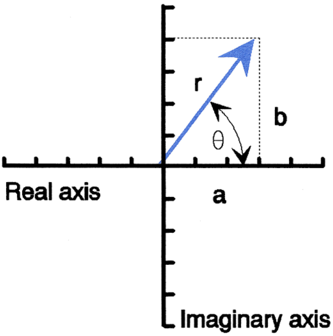
- Press (↱) (POLAR) until the R∠Z or R∠∠ coordinate annunciator is on.

or

- Press (↰) (MODES) (NXT) (NXT) R∠Z or R∠∠.

11 Even though only two coordinate modes are needed for complex numbers, three coordinate modes are available on the HP 48 (to provide for three-dimensional vectors)—Rectangular mode, Cylindrical mode, and Spherical mode. Cylindrical mode (R∠Z) and Spherical mode (R∠∠) are interchangeable for complex numbers—they're both Polar mode.

Complex numbers are displayed inside parentheses. In rectangular form, the real and imaginary parts are separated by a comma. (If the Fraction Mark is set to comma, they're separated by a semicolon instead.) In polar form, the magnitude and phase are separated by a comma and angle sign (∠). (The angle is based on the current angle mode: Degrees, Radians, or Grads.) Regardless of how complex numbers are displayed, the HP 48 stores them internally in rectangular form.



Display Modes

Rectangular	Polar
(a,b)	(r, ∠θ)

Entering Complex Numbers

You can enter complex numbers using either rectangular coordinates or polar coordinates.

To enter a complex number:

- To enter rectangular coordinates, press [↰][( )], enter the coordinates separated by [SPC] or [↰][,], and press [ENTER].
- To enter polar coordinates, press [↰][( )], enter the coordinates separated by [↱][∠], and press [ENTER].
- To use the current coordinate mode, enter the two coordinate values and press [↰][2D] (but only if flag −19 is set). (Don't enter ∠.)

Flag −19 (Complex Mode) specifies whether [↰][2D] (the →V2 command) creates 2D vectors (clear) or complex numbers (set).

The internal rectangular representation of all complex numbers has the following effects on polar numbers:

- θ is normalized to the range ±180° (±π radians, ±200 grads).
- If you key in a negative r, the value is made positive, and θ is increased by 180° and normalized.
- If you key in an r of 0, θ is also reduced to 0.

Example: Enter the number $3 + 4i$ (rectangular coordinates). First, make sure Degrees angle mode and Rectangular coordinate mode are set.

[↰][MODES] [NXT] [NXT]
DEG XYZ

Enter the rectangular complex number.

[↰][( )] 3 [SPC] 4 [ENTER]

Example: Enter the number 5.393 at 158.2 degrees (polar coordinates).

[↰][( )] 5.39 [↱][∠] 158.2

Enter the polar number on the stack. It's converted to match the current coordinate mode (in this case, Rectangular mode).

(ENTER)

```
2:                  (3,4)
1: (-5.00453860689,
    2.00167263362)
DEG ■ RAD GRAD XYZ ■ R∠Z R∠∠
```

11 Now change the coordinate mode and watch how the complex number changes.

(↱)(POLAR)

```
2:  (5,∠53.1301023542)
1:         (5.39,∠158.2)
DEG ■ RAD GRAD XYZ R∠Z ■ R∠∠
```

Press (↱)(POLAR) again to return to Rectangular mode.

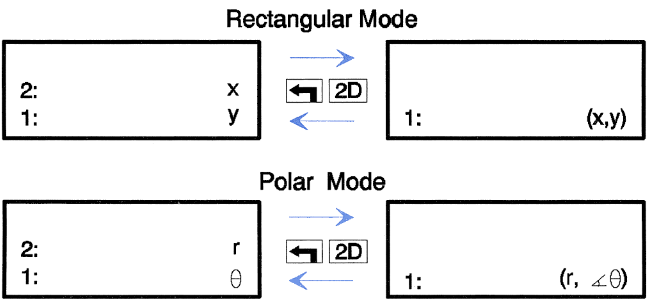
Assembling and Taking Apart Complex Numbers

To assemble a complex number from coordinates on the stack:

- Press (↰)(2D) (but only if flag −19 is set). The coordinates are interpreted according to the current coordinate mode.

To take apart a complex number on the stack:

- Press (↰)(2D). The returned values are the same as the displayed coordinates. (Flag −19 doesn't matter.)



See also the R→C and C→R functions under "Additional Commands for Complex Numbers" on page 11-10.

The programmable equivalents of [⬅][2D] are the →V2 and V→ commands. (See "Additional Commands for Complex Numbers" on page 11-10.)

Example: Assemble the complex number (3,−5) from its components on the stack, then take it apart again. (This example assumes Degrees angle mode and Rectangular coordinate mode are active.)

Set flag −19, then enter the parts on the stack.

19 [+/−] [➡][MODES] [NXT] SF
3 [ENTER] 5 [+/−] [ENTER]

Assemble the complex number.

[⬅][2D]

Take apart the complex number.

[⬅][2D]

Calculating with Complex Numbers

A complex number, like a real number, is a single object. So you can use complex numbers as arguments for commands, and you can use them in symbolic expressions. For symbolic calculations, you can enter a complex number as a coordinate pair inside parentheses *or* as an expression involving the symbolic constant i ($\sqrt{-1}$).

To calculate with complex numbers:

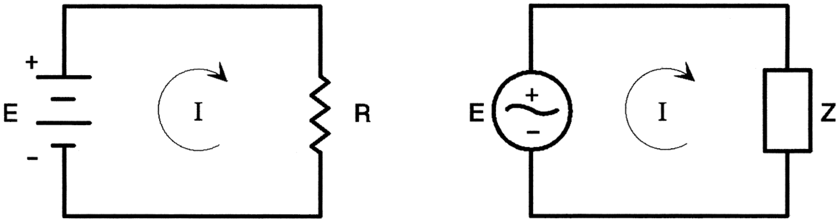
- To use stack syntax, enter the complex numbers, then execute the command.
- To use algebraic syntax, enter a complex number in one of the following ways, then press [ENTER], [EVAL], or [→NUM].
  - To use parentheses, press [⬅][,] between the two coordinates—even if you include ∡.

- To use i, press [α] [←] I for the symbolic constant.

Using Complex Numbers on the Stack

A complex number is a single object—just as a real number is a single object. Most functions that work with real numbers also work with complex numbers.

Example: For the two circuits shown, Ohm's law defines the real-valued resistance R as $R=E\div I$ and the complex-valued impedance Z as $Z=E\div I$. (This example assumes Degrees angle mode is active.)



Given a voltage potential of 10 volts and a current of 2 amperes, calculate the resistance R.

10 [ENTER] 2 [÷]

Given a voltage of (10,∡0) and a current of (2,∡30), calculate the complex impedance. (First, make sure Polar coordinate mode is active.)

[→] [POLAR] (if necessary)
[←] [()] 10 [→] [∡] 0 [ENTER]
[←] [()] 2 [→] [∡] 30 [÷]

Change the coordinate mode to Rectangular.

[→] [POLAR]

In polar form, the complex impedance has a magnitude of 5 and a phase angle of −30 degrees. By changing to the rectangular form, you see that the same complex number implies a resistive component of 4.33 ohms and a reactive component of −2.5 ohms. The negative phase and reactance tell an electrical engineer that the impedance is capacitive, rather than inductive.

11

Example: Calculate

$$\frac{(9+4i)+(-4+3i)}{(3+i)}$$

(This example assumes Degrees angle mode and Rectangular coordinate mode are active.)

Enter the first two complex numbers.

(←)(( )) 9 (SPC) 4 (ENTER) (←)(( )) 4 (+/−) (SPC) 3	1: (9,4) (-4 3◆ TMEN RCLM STOF RCLF SF CF

You do not need to press (ENTER) before pressing (+).

(+)	1: (5,7) TMEN RCLM STOF RCLF SF CF

Divide the result by 3 + i.

(←)(( )) 3 (SPC) 1 (÷)	1: (2.2,1.6) TMEN RCLM STOF RCLF SF CF

Using Complex Numbers in Algebraics

You can represent a complex number in an algebraic in these ways:

- As a coordinate pair inside parenthesis delimiters.
- As an expression involving the symbolic constant i.

The components of a complex number may be real numbers (as in the expression 'X+(1,2)') or they may be formal variables (as in the expression 'X+(A,B)'). Upon entry, this second form is automatically converted to an equivalent form, 'X+(A+B*i)'.

Algebraics containing complex numbers can be manipulated symbolically in the same way as real-number expressions.

Note When you enter a complex number as part of a symbolic expression, you must use [⬑][,] to separate the real and imaginary parts. (If the Fraction Mark is set to comma, [⬑][,] generates a semicolon to separate the parts.)

11

Example: Calculate the sine of (.6,2).

['] [SIN] [⬑][( )] .6 [⬑][,] 2 [EVAL]

```
1: (2.12429548041,
   2.9933770649)
TMEN RCLM STOF RCLF SF CF
```

Example: Calculate the two square roots of the complex number $8-6i$. Because the $\sqrt{}$ function ([$\sqrt{x}$]) returns only *one* root, use the ISOL (isolate) command to solve for W in the equation $W^2 = 8 - 6i$. (The ISOL command is described under "Isolating a Single Variable" on page 22-2.)

First, enter the algebraic.

['] W [y^x] 2 [⬑][=]
[⬑][( )] 8 [⬑][,] 6 [+/-]
[ENTER]

```
1:            'W^2=(8,-6)'
TMEN RCLM STOF RCLF SF CF
```

Enter the name of the variable to be isolated (W) and execute the ISOL command to solve for that variable.

['] W [ENTER]
[⬑][ALGEBRA] ISOL

```
1:           'W=s1*(3,-1)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

The variable s1 stands for ± 1. Thus, the two square roots are $3 - i$ and $-3 + i$.

Example: Use the EquationWriter application to enter an expression representing the complex number $2 - 2i\sqrt{3}$. Then evaluate the expression to get a complex result. (This example assumes Rectangular coordinates mode is active.)

Enter the expression. (Enter "i" as [α] [←] I.)

[←] [EQUATION]
2 [−] 2i [√x] 3

```
2-2·i·√3
COLCT EXPA ISOL QUAD SHOW TAYLR
```


Use the →NUM command to evaluate the expression and return a complex number.

[→] [→NUM]

```
1:   (2,-3.46410161514)
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Real Calculations with Complex Results

The complex-number capabilities of the HP 48 can affect the results of *real*-number operations. Certain calculations that would result in an error on most calculators yield valid complex results on the HP 48. For example, the HP 48 returns a complex number for the square root of −4. Also, the arcsine of 5 yields a complex result.

You'll find that for most calculations, the HP 48 gives you the type of result (real or complex) you expect. However, if you find that you get complex results when you expect real results, check your program or keystrokes for these potential causes:

- The data you supplied to the calculator may be outside the range of the formula you are calculating.
- The formula (or its execution) may be incorrect.
- A rounding error at a critical point in the formula may have compromised the computation.
- A complex result may be unexpected, but correct, for your problem.

Additional Commands for Complex Numbers

Most commands that operate on real numbers also operate on complex numbers (such as SIN, INV, ^, LN, and →Q). The following table describes additional commands that are especially useful for complex numbers.

11

Referring to the table, V→ and →V2 are found in the MTH VECTR menu. NEG is executed in Program-Entry mode by pressing [+/-]. C→R, R→C, and OBJ→ are found in the PRG OBJ menu. The remaining commands are found in the MTH PARTS menu.

Command/Description	Example	
	Input	Output
ABS Absolute value; $\sqrt{x^2+y^2}$.	1: (3,4)	1: 5
ARG Polar angle of a complex number.	1: (1,1)	1: 45
CONJ Complex conjugate of a complex number.	1: (2,3)	1: (2,-3)
C→R Complex to real; separates a complex number into two real numbers, the rectangular coordinates x and y.	1: (2,3)	2: 2 1: 3
IM Imaginary (y) part of a complex number.	1: (4,-3)	1: -3
NEG Negative of its argument.	1: (2,-1)	1: (-2,1)
OBJ→ Object to stack; separates an object (complex number, array, or list) into its elements.	1: (4,5)	2: 4 1: 5

Command/Description	Example	
	Input	Output
RE Real (x) part of a complex number.	1: (4,-3)	1: 4
R→C Real to complex; combines two real numbers into a complex number (x,y).	2: -7 1: -2	1: (-7,-2)
SIGN Unit vector in the direction of the complex number argument; $\left(\frac{x}{\sqrt{x^2+y^2}}, \frac{y}{\sqrt{x^2+y^2}}\right)$	1: (3,4)	1: (.6,.8)
V→ Separates a complex number into two real numbers x and y or r and θ, depending on the current coordinates mode. The example assumes Polar and Degrees modes.	1: (3,∡30)	2: 3 1: 30
→V2 If flag −19 is set, assembles two real numbers into a complex number (x,y) or (r,$\measuredangle\theta$), depending on the current coordinate mode. The example assumes Polar and Degrees modes.	2: 6 1: 50	1: (6,∡50)

11

Choosing Complex Numbers or Vectors

Complex numbers and two-dimensional vectors can be similar in many ways. Sometimes you may have difficulty choosing the better object type to use for a given problem (and sometimes either type will work).

11

The main advantages of using complex numbers are that they're allowed as elements of vectors and matrices and that most real-number operations work on them. The main advantages of using vectors are that they're not limited to two dimensions and that vector operations like DOT and CROSS work on them.

If you make the wrong choice at the start of a calculation, you can convert from one type to the other.

To convert to a 2D vector or to a complex number:

- To take apart a complex number in level 1 and reassemble the parts as a vector, clear flag −19, then press [◄][2D] [◄][2D].
- To take apart a 2D vector in level 1 and reassemble the parts as a complex number, set flag −19, then press [◄][2D] [◄][2D].

Example: Convert the complex number (3,4) into a vector. (This example assumes Rectangular and Degrees modes are active.)

Enter the complex number.

[◄][( )] 3 [SPC] 4 [ENTER]

```
1:                    (3,4)
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Clear flag −19 so that [◄][2D] assembles a vector.

19 [+/-] [►][MODES]
[NXT] CF

```
1:                    (3,4)
TMEN RCLM STOF RCLF  SF   CF
```

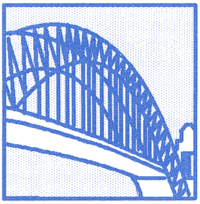
Take apart the complex number and reassemble the parts into a vector.

[◄][2D] [◄][2D]

```
1:                  [ 3 4 ]
TMEN RCLM STOF RCLF  SF   CF
```

12

Vectors



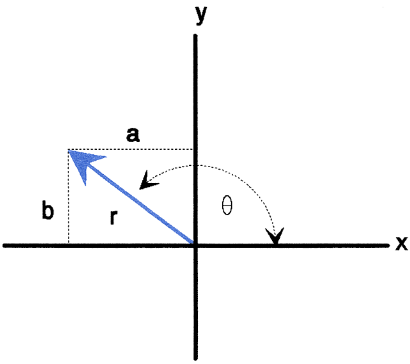
The HP 48 has special facilities for working with 2-dimensional (2D) and 3-dimensional (3D) vectors.

All vectors are array objects. The general case of n-dimensional vectors is covered in chapter 20, "Arrays"—this chapter deals primarily with 2D and 3D vectors. It covers the following topics:

- Interpreting and controlling the display format of vectors.
- Entering vectors.
- Assembling and taking apart vectors.
- Calculating with vectors.
- Determining when to use complex numbers and when to use vectors.

Displaying 2D and 3D Vectors

You can display 2D vectors as either rectangular components ([*X* *Y*]) or polar components ([*R* ∡])—in *Rectangular mode* or in *Polar mode*.

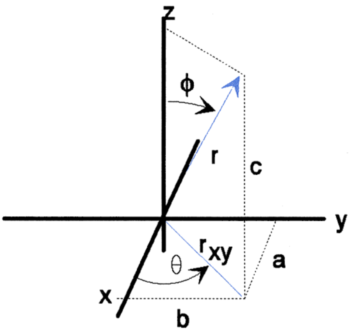


Two-Dimensional Display Modes

Rectangular	Polar
[a b]	[r ∡θ]

2D Vector Components

You can display 3D vectors as rectangular components (`[ X Y Z ]`), cylindrical components (`[ R ∡ Z ]`), or spherical components (`[ R ∡ ∡ ]`)—in *Rectangular mode*, in *Cylindrical mode*, or in *Spherical mode*.



Three-Dimensional Display Modes

Rectangular	Cylindrical	Spherical
[a b c]	[r_{xy} ∡θ c]	[r ∡θ ∡ϕ]

3D Vector Components

Polar mode is actually two modes—Cylindrical mode and Spherical mode. For 2D vectors, Cylindrical and Spherical modes are interchangeable—both give the same two-dimensional results.

To display rectangular components:

- Press (↱)(POLAR) until *no* coordinate annunciator is on.

  or
- Press (MTH) VECTR XYZ .

To display polar (cylindrical or spherical) components:

12

- Press (↱)(POLAR) until the R∡Z or R∡∡ coordinate annunciator is on.

  or
- Press (MTH) VECTR R∡Z (for cylindrical/polar) or R∡∡ (for spherical/polar).

You can also change the coordinate mode in page 3 of the MODES menu ((↰)(MODES) (NXT) (NXT)).

The ■ in the menu label and the coordinate annunciator indicate the active coordinate mode:

- Rectangular mode: XYZ■, no annunciator.
- Cylindrical mode: R∡Z■, R∡Z annunciator.
- Spherical mode: R∡∡■, R∡∡ annunciator.

Vectors are displayed inside [ ] delimiters. In rectangular form, the components are separated by spaces. In polar (cylindrical or spherical) form, angles are preceded by an angle sign (∡). (The angle is based on the current angle mode: Degrees, Radians, or Grads.) Regardless of how vectors are displayed, the HP 48 stores them internally in rectangular form.

If you want to analyze a right triangle, you can use the rectangular coordinates of a vector to represent the sides of the triangle, and the polar coorinates to represent the hypotenuse and one angle of the triangle. If you enter one type of coordinates, you can simply change the coordinate mode to see the other parameters.

Entering 2D and 3D Vectors

You can enter 2D and 3D vectors using any type of components: rectangular, cylindrical/polar, or spherical/polar.

To enter a 2D or 3D vector:

12

- To enter a specific type of components, press [⬅][[ ]], enter the components separated by [SPC] or [➡][∠], and press [ENTER]. (Press [➡][∠] just before each angular component.)
- To use the current coordinate mode, enter the two or three component values and press [⬅][2D] or [➡][3D] (but use [⬅][2D] only if flag −19 is clear). (Don't enter ∡.)

Flag −19 (Complex Mode) specifies whether [⬅][2D] (the →V2 command) creates 2D vectors (clear) or complex numbers (set).

The internal rectangular representation of all vectors has the following effects on displayed polar (cylindrical and spherical) vectors:

- θ is normalized to within ±180° (±π radians, ±200 grads).
- ϕ is normalized to within 0 to 180° (0 to π radians, 0 to 200 grads).
- If you key in a negative r, the value is made positive; θ is increased by 180°, ϕ by 90°, and both are normalized.
- If ϕ is 0° or 180°, θ is reduced to 0°.
- If you key in an r of 0, θ and ϕ are reduced to 0°.

Example: Enter the vector [3 4] (rectangular components). First, make sure Degrees mode and Rectangular mode are set.

[⬅][RAD] (if necessary)
[MTH] VECTR XYZ

```
XYZ■ R∠Z R∠∠ CROSS DOT ABS
```

Enter the rectangular vector.

[⬅][[ ]] 3 [SPC] 4 [ENTER]

```
1:                  [ 3 4 ]
XYZ■ R∠Z R∠∠ CROSS DOT ABS
```

Example: Entry the vector 5.39 at 158.2 degrees (polar components). (When entering polar vectors, you don't need a space to separate the elements—the angle sign acts as the separator.)

[◄] [[ ]] 5.39 [►] [∠] 158.2

```
1:                [ 3 4 ]
[5.39∠158.2◆
 XYZ■ R∠Z  R∠∠  CROSS DOT  ABS
```

Enter the polar vector on the stack. It's converted to match the current display mode (in this case, Rectangular mode).

[ENTER]

```
2:                [ 3 4 ]
1: [ -5.00453860689
     2.00167263362 ]
 XYZ■ R∠Z  R∠∠  CROSS DOT  ABS
```

Now change the coordinate mode and watch how the vectors change.

[►] [POLAR]

```
2: [ 5 ∠53.1301023542…
1:       [ 5.39 ∠158.2 ]
 XYZ  R∠Z■ R∠∠  CROSS DOT  ABS
```

Press [►] [POLAR] again to return to Rectangular mode.

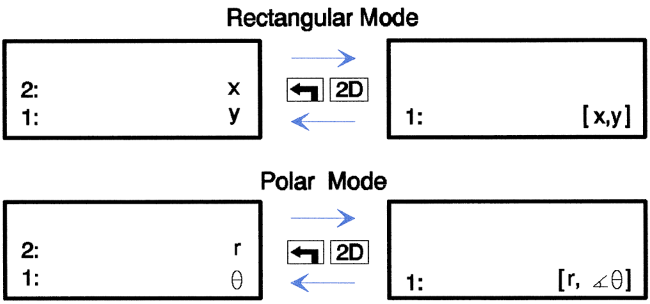
Assembling and Taking Apart 2D and 3D Vectors

To assemble a 2D or 3D vector from components on the stack:

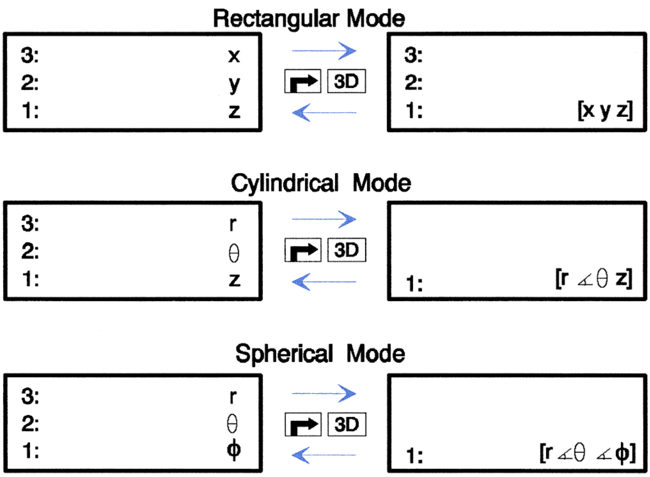
- For two components, press [◄] [2D] (but only if flag −19 is clear). The components are interpreted according to the current coordinate mode.
- For three components, press [►] [3D]. The components are interpreted according to the current coordinate mode.

To take apart a 2D or 3D vector on the stack:

- For a 2D vector, press [◄] [2D]. The returned values are the same as the displayed components. (Flag −19 doesn't matter.)
- For a 3D vector, press [►] [3D]. The returned values are the same as the displayed components.



Assembling and Taking Apart 2D Vectors



Assembling and Taking Apart 3D Vectors

The programmable equivalents of ↰ 2D are V→ and →V2, and the programmable equivalents of ↱ 3D are the V→ and →V3 commands. (See "Additional Vector Commands" on page 12-14.)

Example: 2D Vector. Assemble the two-dimensional vector [3 5] from its components on the stack, then take it apart again. (This example assumes Rectangular coordinate mode is active and flag −19 is clear.)

Enter the real-number components.

3 (ENTER) 5 (ENTER)

Assemble the vector.

(◄)(2D)

Separate the vector into its components.

(◄)(2D)

Example: 3D Vector. Assemble the three-dimensional vector [10 ∡240 ∡20] from its components, then take it apart again. (This example assumes Degrees mode is active.)

Set Spherical coordinate mode, then enter the real numbers associated with the vector.

(MTH) VECTR R∠∠
10 (SPC) 240 (SPC) 20 (ENTER)

Assemble the vector. (Note that 240° is converted to −120° when the angles are normalized.)

(►)(3D)

Take the vector apart.

(►)(3D)

Press (►)(POLAR) to return to Rectangular mode.

Calculating with 2D and 3D Vectors

A vector, like a real number, is a single object. So you can use vectors as arguments for commands. You can add and subtract vectors—you can multiply and divide vectors by scalars—and you can execute special vector commands (DOT, CROSS, and ABS) with them. (The
12 absolute value function ABS returns the magnitude of a vector.)

To calculate with vectors:

- Enter the vectors on the stack, then execute the command.

Example: Finding the Unit Vector. A unit vector parallel to a given vector is found by dividing a vector by its magnitude. Find the unit vector for [3 4 5]. (This example assumes Rectangular coordinate mode and Degrees angle mode are active).

Enter the vector.

3 (ENTER) 4 (ENTER) 5 (↱)(3D)

```
1:                [ 3 4 5 ]
XYZ■ R∠Z  R∠∠  CROSS DOT  ABS
```

Duplicate the vector and compute its magnitude.

(ENTER) ABS

```
2:                [ 3 4 5 ]
1:            7.07106781187
XYZ■ R∠Z  R∠∠  CROSS DOT  ABS
```

Divide the vector by its magnitude to get the unit vector.

(÷)

```
1: [ .424264068712
     .565685424949
     .707106781186 ]
XYZ■ R∠Z  R∠∠  CROSS DOT  ABS
```

Example: Finding the Angle between Vectors. The angle between two vectors is given by

$$angle = \cos^{-1}\left[\frac{\mathbf{V1}\cdot\mathbf{V2}}{|\mathbf{V1}|\,|\mathbf{V2}|}\right]$$

Calculate the angle between the vectors [3 4 5] and [20 ∡30 ∡60]. (This example assumes Rectangular and degrees modes are active.)

Enter both vectors. (Notice the change to Spherical mode for entering the second vector.)

3 [ENTER] 4 [ENTER] 5 [→] [3D]
[MTH] VECTR R∠∠
20 [ENTER] 30 [ENTER] 60 [→] [3D]

```
2: [ 7.07106781187 ∠5…
1:       [ 20 ∠30 ∠60 ]
XYZ  R∠Z  R∠∠■ CROSS DOT ABS
```

Take the dot product.

DOT

```
1:        129.641016151
XYZ  R∠Z  R∠∠■ CROSS DOT ABS
```

Return the vectors to the stack.

[→] [LAST ARG]

```
3:        129.641016151
2: [ 7.07106781187 ∠5…
1:       [ 20 ∠30 ∠60 ]
XYZ  R∠Z  R∠∠■ CROSS DOT ABS
```

Use ABS to find the magnitude of each vector.

ABS [←] [SWAP] ABS

```
3:        129.641016151
2:                   20
1:        7.07106781187
XYZ  R∠Z  R∠∠■ CROSS DOT ABS
```

Multiply the magnitudes and divide the result into the dot product.

[×] [÷]

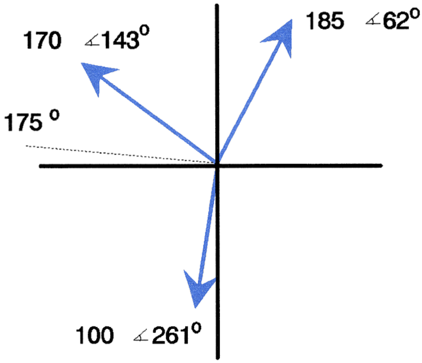
```
1:        .916700416405
XYZ  R∠Z  R∠∠■ CROSS DOT ABS
```

Use ACOS to find the angle.

[←] [ACOS]

```
1:        23.5516253446
XYZ  R∠Z  R∠∠■ CROSS DOT ABS
```

Example: Finding the Component in One Direction. The following diagram represents three 2D vectors. Find their sum, and then use DOT to resolve them along the 175° line. (This example assumes Degrees mode is active.)



Set Polar mode (Cylindrical mode in this case), then enter the three vectors.

(MTH) VECTR R∡Z
170 (ENTER) 143 (↰)(2D)
185 (ENTER) 62 (↰)(2D)
100 (ENTER) 261 (↰)(2D)

```
3:          [ 170 ∡143 ]
2:           [ 185 ∡62 ]
1:          [ 100 ∡-99 ]
 XYZ  R∡Z■ R∡∡ CROSS DOT ABS
```

Add them.

(+)(+)

```
1: [ 178.937160532
   ∡111.148894255 ]
 XYZ  R∡Z■ R∡∡ CROSS DOT ABS
```

Enter the unit vector of 175°.

1 (ENTER) 175 (↰)(2D)

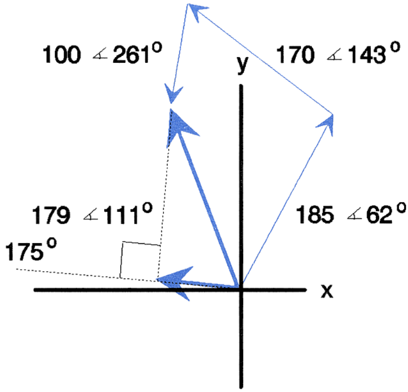
```
2: [ 178.937160532 ∡1…
1:            [ 1 ∡175 ]
 XYZ  R∡Z■ R∡∡ CROSS DOT ABS
```

Find the scalar representing the magnitude of the force along the 175° line.

DOT

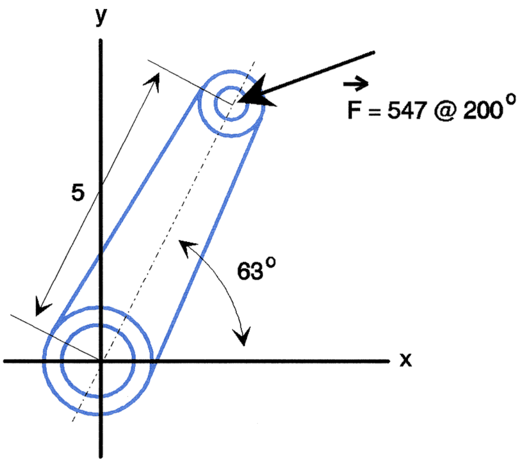
```
1:         78.8585649505
 XYZ  R∡Z■ R∡∡ CROSS DOT ABS
```

This illustration shows how the vectors add and then resolve in the given direction.



12

Example: Taking Vector Cross Products. For the crank below, find the moment about the origin, and find the force transferred along the axis of the crank.



The moment is found by taking the vector cross product of the crank radius and force vectors. To take a cross product, enter the vectors in the same order that they appear in the cross-product formula: $\mathbf{M} = \mathbf{r} \times \mathbf{F}$. (This example assumes Degrees mode is active.)

Set Polar mode and enter the radius and force vectors.

(MTH) VECTR R∠∠
5 (ENTER) 63 (↰)(2D)
547 (ENTER) 200 (↰)(2D)

```
2:            [ 5 ∠63 ]
1:        [ 547 ∠-160 ]
 XYZ  R∠Z  R∠∠■ CROSS  DOT  ABS
```

12 Take the cross product. (Notice that the result is a three-dimensional vector.)

CROSS

```
1: [ 1865.26551477 ∠0
   ∠0 ]
 XYZ  R∠Z  R∠∠■ CROSS  DOT  ABS
```

You would expect this three-dimensional vector to be positive and parallel to the z axis. This can be verified by inspection and the right hand rule. Change to Rectangular mode to make the verification easier.

(↱)(POLAR)

```
1: [ 0 0 1865.26551477
   ]
 XYZ■ R∠Z  R∠∠ CROSS  DOT  ABS
```

Now return the original vectors to the stack and change back to Polar mode.

(↱)(LAST ARG) (↱)(POLAR)

```
3: [ 1865.26551477 ∠0…
2:            [ 5 ∠63 ]
1:        [ 547 ∠-160 ]
 XYZ  R∠Z  R∠∠■ CROSS  DOT  ABS
```

Duplicate the radius and divide it by its magnitude to get the unit vector.

(▶) (ENTER) ABS (÷)

```
3: [ 1865.26551477 ∠0…
2:        [ 547 ∠-160 ]
1:            [ 1 ∠63 ]
 XYZ  R∠Z  R∠∠■ CROSS  DOT  ABS
```

Take the dot product to find the scalar representing the magnitude of the force along the crank.

DOT

```
2: [ 1865.26551477 ∠0…
1:      -400.050474786
 XYZ  R∠Z  R∠∠■ CROSS  DOT  ABS
```

The negative magnitude indicates that the force is opposed to the direction of the crank's unit vector. (See the next example.)

Example: Continuation. To add a small twist to the previous example, suppose the force vector is not on the same plane as the crank. If the force is [547 ∡200 ∡87] (thus the force vector rises out of the paper at a modest 3°), find the moment, the force transmitted along the axis of the crank, and the thrust force along the z axis.

Enter the radius and force vectors. (Use Cylindrical mode with a z-value of 0 for the radius vector, and use Spherical mode for the force.)

R∡Z
5 (ENTER) 63 (ENTER) 0 (↱)(3D)
R∡∡
547 (ENTER) 200 (ENTER) 87 (↱)(3D)

```
2:        [ 5 ∡63 ∡90 ]
1:   [ 547 ∡-160 ∡87 ]
 XYZ | R∡Z | R∡∡■|CROSS| DOT | ABS
```

Take the cross product.

CROSS

```
1: [ 1868.20084977
   ∡-27 ∡4.39422603566
   ]
 XYZ | R∡Z | R∡∡■|CROSS| DOT | ABS
```

This time the resulting moment is not directed precisely along the z axis. Switch to Rectangular mode and see the z-axis component. (The useful moment along the crank has a magnitude of almost 1863.)

XYZ

```
1: [ 127.537640594
   -64.9836736511
   1862.70923321 ]
 XYZ■| R∡Z | R∡∡ |CROSS| DOT | ABS
```

Now get the original vectors back on the stack. Notice that the thrust problem has been solved through the switch to Rectangular mode. (The thrust—the z-component of the vector—is approximately 28.6. Note that it is positive and comes out of the paper the same as the force vector. The same value could have been calculated through a more general approach of calculating the dot product of the unit vector associated with the z axis [0 0 1] and the force vector.)

(↱)(LAST ARG)

```
2: [ 2.2699524987 4.4…
1: [ -513.307428175
   -186.828624883
   28.6277680649 ]
 XYZ■| R∡Z | R∡∡ |CROSS| DOT | ABS
```

Compute the force along the crank.

(▶) (ENTER)
ABS (÷) DOT

```
2: [ 127.537640594 -6…
1:        -399.502219513
 XYZ■| R∡Z | R∡∡ |CROSS| DOT | ABS
```

Additional Vector Commands

The following commands interpret their arguments and return results using the current coordinate mode. These commands are found on page 2 of the MTH VECTR menu ((MTH) VECTR (NXT)).

12

Command/Description	Example	
	Input	Output

V→ Separates a vector (or complex number) into its component elements according to the current coordinate mode. The examples assume Degrees mode.	`1: [8 9]`
	
	`1: [5 ∡90 3]`
		`2: 8`
		`1: 9`
		
		`3: 5`
		`2: 90`
		`1: 3`

→V2 If flag −19 is clear, assembles two real numbers into a 2-element vector according to the current coordinate mode. The example assumes Polar and Degrees modes.	`2: 2` `1: 20`	`1: [2 ∡20]`

→V3 Assembles three real numbers into a 3-element vector according to the current coordinate mode. The example assumes Spherical and Degrees modes.	`3: 2`
	`2: 20`
	`1: 5`
		`1:[2 ∡20 ∡5]`

Additional commands for manipulating vectors are →ARRY, GET, GETI, OBJ→, PUT, and PUTI. These are covered in the table under "Manipulating Objects" on page 4-12.

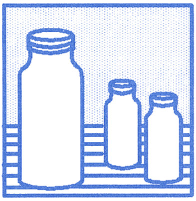
Choosing Complex Numbers or Vectors

Complex numbers and two-dimensional vectors can be similar in many ways. Sometimes you may have difficulty choosing the better object type to use for a given problem (and sometimes either type will work).

See "Choosing Complex Numbers or Vectors" on page 11-12 for a comparison and example.

13

Unit Management



The Units application contains a catalog of 147 units that you can combine with real numbers to create *unit objects*. The Units application lets you:

- Convert units. For example, you can convert the unit object 10_ft to 120_in or 3.048_m.
- Factor units. For example, you can factor 20_W with respect to 1_N and return 20_N*m/s.
- Calculate with units. For example, you can add 10_ft/s to 10_mph and return 24.67_ft/s.

Overview of the Units Application

The Units application consists of two menus:

- The UNITS Catalog menu ([←][UNITS]), which contains all the HP 48 units, organized by subject. You use the UNITS Catalog menu to create unit objects and to convert between related units in the catalog.
- The UNITS Command menu ([→][UNITS]), which contains commands for converting units and for managing unit objects.

Units and Unit Objects

The Units application is based on the International System of Units (SI). The International System specifies seven *base* units: m (meter), kg (kilogram), s (second), A (ampere), K (kelvins), cd (candela), and mol (mole). The UNITS Catalog menu contains the seven base units and 141 *compound* units derived from the base units. For example, in (inch) is .0254 m, and Fdy (Faraday) is 96487 A*s.

A *unit object* has two parts: a *number* (a real number) and a *unit expression* (a single unit or multiplicative combination of units). The two parts are linked by the _ character. For example, 2_in (2 inches), X*1_N (*X* Newtons), and 8.303_gal/h (8.303 US gallons per hour) are unit objects. Like other object types, a unit object can be placed on the stack, stored in a variable, and used in algebraic expressions and programs.

When you perform a *unit conversion*, the HP 48 replaces the old unit expression with a new unit expression (specified by you), and automatically multiplies the number by the appropriate conversion factor.

The UNITS Catalog Menu

The UNITS Catalog menu ((⬅)(UNITS)) displays a three-page menu of "subject" keys, each of which, when pressed, displays a submenu of related units. For example, (⬅)(UNITS) (NXT) PRESS displays a two-page menu of units for pressure.

The individual keys in each submenu behave differently from standard menu keys, as described throughout this chapter. In Immediate-entry mode, when you press the

- Unshifted key, the HP 48 *creates* a unit object that corresponds to that key. (In Algebraic- or Program-entry modes, the unshifted keys act as typing aids, echoing the corresponding unit name into the command line.)
- Left-shifted key, the HP 48 *converts* the unit object in the command line or stack level 1 to the corresponding unit.
- Right-shifted key, the HP 48 *divides by* the corresponding unit. This helps you create unit expressions with units in the denominator.

The use of the UNITS Catalog menu is discussed in detail in this chapter.

Creating a Unit Object

The UNITS Catalog menu provides a simple method for creating a unit object.

To create a unit object on the stack:

1. Key in the number part of the unit object.
2. Press [←] [UNITS] and select the subject menu that contains the desired unit.
3. Press the menu key for the unit you want. (If you want the *inverse* of the unit, press [→] and the menu key.)
4. For compound units, repeat steps 2 through 4 for each individual unit in the unit expression.

When you press a menu key in the UNITS Catalog menu, the HP 48 first enters a corresponding unit object on the stack with the number value 1. Then, for an unshifted key, it executes * (multiply)—or for a right-shifted key, it executes / (divide).

Example: Create the unit object 3.5_ft^3.

Select the VOL submenu of the UNITS Catalog menu.

[←] [UNITS] VOL

Key in the number, then append the unit.

3.5 FT^3

Example: Create the unit object 32_kg*m^2/s^2.

Key in the number and append the first unit.

32 [←] [UNITS] MASS KG

Append the second unit.

⬅ UNITS AREA M^2

```
1:              32_kg*m^2
M^2 CM^2 B YD^2 FT^2 IN^2
```

Append the units in the denominator.

⬅ UNITS TIME
13 ➡ S ➡ S

```
1:          32_kg*m^2/s^2
YR D H MIN S HZ
```

To create a unit object in the command line:

1. Key in the number.
2. Key in the _ character (press ➡ _). This activates Algebraic-entry mode.
3. Key in the unit expression as you would an algebraic expression:
   - To key in a unit name, either press the corresponding menu key or spell the unit name.
   - To create compound units, press ×, ÷, y^x, and ⬅ ( ) as required.

Note that unit names are case-sensitive. For example, Hz (hertz) must be typed with uppercase H and lowercase z. (For legibility, all letters in menu keys are uppercase. Don't confuse the menu-key representation of a unit with its proper name.)

By spelling unit names, you can create a unit object without switching between submenus in the UNITS Catalog menu. However, the menu keys eliminate errors resulting from incorrect spelling and incorrect use of uppercase or lowercase.

Example: Create the unit object 8_Btu/(ft^2*h*°F) in the command line.

Key in the number and the _ character. Then key in the unit expression using alpha characters. (To type °, press α ➡ 6.) Then enter the unit object.

8 ➡ _
Btu ÷ ⬅ ( ) ft y^x 2
× h × °F
ENTER

```
1:     8_Btu/(ft^2*h*°F)
YR D H MIN S HZ
```

To create a unit object using the EquationWriter application:

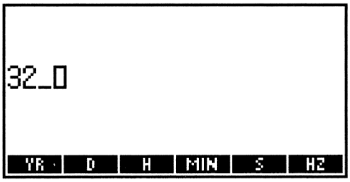
1. Press (←)(EQUATION).
2. Enter the number, press (→)(_), and enter the unit expression using standard EquationWriter notation.
3. Press (ENTER).

The EquationWriter application lets you build algebraics that contain unit objects, showing you the unit expression as you would write it on paper. Inverse units are displayed in fractional form, and exponents are displayed as superscripts. (See "Entering Equations" on page 16-5 for details about using the EquationWriter application and unit objects.)

Example: Use the EquationWriter application to create the unit object 32 W/m^2*°C. (This procedure is explained on page 16-10.)

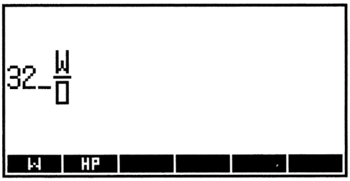
Select the EquationWriter application. Key in the number and start the unit expression.

(←)(EQUATION)
32 (→)(_)



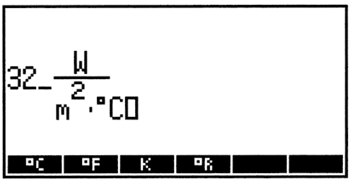
Key in the numerator of the unit expression.

(▲) (←)(UNITS) (NXT) POWR
W (▶)



Key in the denominator.

(←)(UNITS) AREA M^2 (▶)
(←)(UNITS) (NXT) TEMP °C



Put the unit object on the stack.

(ENTER)

To view a unit object using the EquationWriter application:

- Press [▼] while the object is in level 1.

To check the spelling and case for a unit:

1. Press [⬅][UNITS] and select the corresponding page in the menu.
2. Press [⬅][REVIEW]. A temporary display lists each unit on that menu page.
3. Press [ATTN] to return to the stack.

Example: Check the correct spelling and case for the unit corresponding to the FT*LB key in the UNITS ENRG submenu.

[⬅][UNITS] [NXT] ENRG
[⬅][REVIEW]

Press [ATTN] to return to the stack display.

Operators in unit objects follow this precedence order:

1. ( ) (highest precedence).
2. ^.
3. * and /.

For example, 7_m/s^2 is 7 meters per second, and 7_(m/s)^2 is 7 square meters per square second.

You can also insert a unit prefix in front of a unit. Unit prefixes are letters that indicate powers of ten. For example, mA means “milliamp” (amp $\times$ 10^{-3}). The following table lists allowable prefixes. (To key in μ, press [α] [➡] N.)

Unit Prefixes

Prefix	Name	Exponent	Prefix	Name	Exponent
E	exa	+18	d	deci	−1
P	peta	+15	c	cent	−2
T	tera	+12	m	milli	−3
G	giga	+9	μ	micro	−6
M	mega	+6	n	nano	−9
k *or* K	kilo	+3	p	pico	−12
h *or* H	hecto	+2	f	femto	−15
D	deka	+1	a	atto	−18

Most prefixes used by the HP 48 correspond with standard SI notation—with one exception: "deka" is "D" in HP 48 notation and "da" in SI notation.

Note

You cannot use a prefix with a built-in unit if the resulting unit matches another built-in unit. For example, you cannot use `min` to indicate milli-inches, because `min` is a built-in unit indicating "minutes." Other possible combinations that match built-in units are `Pa`, `da`, `cd`, `ph`, `flam`, `nmi`, `mph`, `kph`, `ct`, `pt`, `ft`, `au`, and `cu`.

Using Unit Objects in Algebraics

Unit objects are allowed in algebraics—you enter them just as you enter them in the command line. In addition, the command line permits symbolic numbers instead of real numbers, converting `'Y_ft'`, for example, to `Y*1_ft` when entered on the stack.

+ and − are allowed in the number. However, the _ character takes precedence over + and −. Thus `'(4+5)_ft' EVAL` returns `9_ft`, but `'4+5_ft' EVAL` returns `+ Error: Inconsistent Units`.

Converting Units

You can convert unit objects to different units using four methods:

- The UNITS Catalog menu. Converts to built-in units only.
- The CONVERT command. Converts to any units.
- The CST (custom) menu. Converts to any units already set up.
- The UBASE (base units) command. Converts to SI base units only.

If you're working with temperature units, see "Working with Temperature Units" on page 13-17.

Using the UNITS Catalog Menu

The UNITS Catalog menu lets you convert the unit object in stack level 1 to any dimensionally consistent unit in the menu.

To convert units to a built-in unit:

1. Enter the unit object with the original units.
2. Press [←] [UNITS] and select the subject menu that contains the desired unit.
3. Press [←] and the menu key for the desired unit.

Example: Convert `10_atm` (atmospheres) to `inHg` (inches of mercury). First, create the unit object `10_atm`.

[←] [UNITS]
[NXT] PRESS
10 ATM

Convert to inches of mercury.

[NXT] [←] INHG

Example: Convert `6_ft*lbf/s` (foot-pound force per second) to `W` (watts). First, enter the unit object.

6 [→] [_] [←] [UNITS] [NXT]
ENRG FT*LB
[←] [UNITS] TIME [→] S

Select the POWR submenu and convert to watts.

(⬅)(UNITS) (NXT) POWR
(⬅) W

```
1:      8.13490768999_W
 W  | HP |    |    |    |
```

Using CONVERT

You can use the CONVERT command to do *any* conversion between dimensionally consistent unit expressions.

To convert to any units:

1. Enter the unit object with the original units.
2. Enter any number (such as 1) and attach the units you want to convert to.
3. Press (➡)(UNITS) CONV.

CONVERT converts the level 2 unit object using the units from the level 1 object. It ignores the number part of the level 1 unit object.

Example: Convert 12_ft^3/min (cubic feet per minute) to qt/h (quarts per hour). Since qt/h is not in the UNITS Catalog menu, you must use CONVERT to do the conversion.

Enter the unit object.

12 (⬅)(UNITS) VOL FT^3
(⬅)(UNITS) TIME (➡) MIN

```
1:          12_ft^3/min
 YR | D | H | MIN | S | HZ
```

Put the new unit expression on the stack, appended to the number 1. (The number is ignored.)

1 (➡)(LAST MENU) (NXT) QT
(➡)(LAST MENU) (➡) H

```
2:          12_ft^3/min
1:               1_qt/h
 YR | D | H | MIN | S | HZ
```

Perform the conversion.

(➡)(UNITS) CONV

```
1:   21543.8961039_qt/h
CONV|UBASE|UVAL|UFACT|→UNIT|
```

(Note how you can use (LAST MENU) to bypass the main UNITS Catalog menu and directly select the previous submenu.)

Using the CST Menu

If you often execute a specific unit conversion, you may find it convenient to execute that conversion from the CST (custom) menu, particularly if the unit expressions are not in the UNITS Catalog menu.

To set up the CST menu for unit operations:

- Include a unit object with the desired units in the CST menu list—the number part of the object is ignored. (See "Creating a Custom Menu" on page 15-1 for details.)

Unit keys in the CST menu operate the same as keys in the UNITS menus.

To enter units from the CST menu:

- Press [CST] and the menu key for the desired unit. (If you want the *inverse* of the unit, press [→] and the menu key.)

To convert units to CST menu units:

1. Enter the unit object with the original units.
2. Press [CST].
3. Press [←] and the menu key for the desired unit.

Example: Suppose you often execute unit conversions between `kg/m^3` (kilograms per cubic meter) and `lb/ft^3` (pounds per cubic foot). Put those unit expressions in the CST menu. (This example assumes there are no other entries in the CST menu list.)

Build a list that contains the two unit objects. (When you press [←][{}], the HP 48 switches to Program-entry mode, so you have to key in the _ and / characters.)

[←][{}]
1 [→][_] [←][UNITS] MASS KG
[÷] [←][UNITS] VOL M^3 [SPC]
1 [→][_] [→][LAST MENU] LB [÷]
[→][LAST MENU] FT^3 [ENTER]

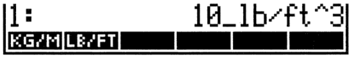
```
1: { 1_kg/m^3 1_lb/ft^
   3 }
M^3 | ST | CM^3 | YD^3 | FT^3 | IN^3
```

Store the list in the variable *CST* and display the CST menu. (This procedure is explained on page 15-1.)

(→) (MODES) MENU

Now convert 10_lb/ft^3 to kg/m^3. Enter the unit object.

10 LB/FT



Convert to kilograms per cubic meter.

(←) KG/M

Using UBASE (for SI Base Units)

The UBASE command converts a compound unit into its equivalent SI base units.

To convert units to SI base units:

1. Enter the unit object with the original units.
2. Press (→) (UNITS) UBASE.

Example: Convert 8.3_Pa (Pascals) into SI base units.

(←) (UNITS) (NXT) PRESS
8.3 PA
(→) (UNITS) UBASE

Example: Convert 30_knot into SI base units.

(←) (UNITS) SPEED
30 KNOT
(→) (UNITS) UBASE

13

Converting Dimensionless Units of Angle

Plane and solid angles are *dimensionless*. You can use the following dimensionless units as constants in your unit expressions; however the HP 48 can't check for dimensional consistency in dimensionless units.

Dimensionless Unit	Unit Name	Value
Arcmin	arcmin	$^1/_{21600}$ unit circle
Arcsec	arcs	$^1/_{1296000}$ unit circle
Degree	°	$^1/_{360}$ unit circle
Grad	grad	$^1/_{400}$ unit circle
Radian	r	$^1/_{2\pi}$ unit circle
Steradian	sr	$^1/_{4\pi}$ unit sphere

Some photometric units are defined in terms of steradians. These units include a factor of $^1/_{4\pi}$ in their numeric values. Because this factor is dimensionless, the HP 48 can't check for its presence or absence—it can't check that your units are consistent. The following table lists photometric units according to whether their definition includes steradians.

Include Steradians	Do Not Include Steradians
Lumen (lm)	Candela (cd)
Lux (lx)	Footlambert (flam)
Phot (ph)	Lambert (lam)
Footcandle (fc)	Stilb (sb)

To convert photometric units:

- To convert between photometric units in the same column, the sr unit is *not* required.
- To convert between photometric units in different columns, divide the unit in the left column by sr or multiply the unit in the right column by sr.

Some examples of consistent photometric units are:

- lm is consistent with cd*sr.
- fc/sr is consistent with flam.
- lm/sr*m^2 is consistent with lam.

Factoring Unit Expressions

The UFACT command factors one unit within a unit expression, returning a unit object whose unit expression consists of the factored unit and the remaining SI base units.

To factor units within a unit expression:

1. Enter the unit object with the original units.
2. Enter any number (such as 1) and attach the units you want to factor out.
3. Press (↱)(UNITS) UFACT.

UFACT factors the units of the level 1 object from the level 2 unit object.

Example: Factor 3.5_kg*m^2/s^2 with respect to N (Newtons). First, enter the unit object.

(↰)(UNITS) MASS 3.5 KG
(↰)(UNITS) AREA M^2
(↰)(UNITS) TIME
(↱) S (↱) S

Key in the unit to be factored.

1 (↰)(UNITS)(NXT) FORCE N

Factor the level 2 unit object.

(↱)(UNITS) UFACT

13

Calculating with Units

The HP 48 lets you execute many arithmetic operations with unit objects, just as you execute them with real numbers:

- Addition and subtraction (dimensionally consistent units only).
- Multiplication and division.
- Inversion.
- Raising to a power.
- Percentage calculations (dimensionally consistent units only).
- Comparisons of values (dimensionally consistent units only).
- Trigonometric operations (planar angular units only).

Several additional math operations work only on the number part of the unit object.

To calculate with unit objects:

1. Enter the unit objects.
2. Execute the commands.

Units are automatically converted and combined during the calculation. Certain operations require dimensionally consistent units—units that have the same physical dimensions, such as length or density. For such operations, results with units are converted to the units from the object in level 1.

Temperature units require special note—see "Working with Temperature Units" on page 13-17.

The trigonometric operations SIN, COS, and TAN on unit objects require a *planar angular* unit. Planar angular units are radians (r), degrees (°), grads (grad), arc-minutes (arcmin), and arc-seconds (arcs). The result is a dimensionless real number.

The following functions, described in detail under "Other Real-Number Functions" on page 9-14, operate on the number part of a unit object. Each function returns a unit object, leaving the unit-expression part of the argument unchanged:

ABS	FLOOR	IP	RND
CEIL	FP	NEG	TRNC

The SIGN function, described in the same section, returns a *number* that indicates the sign of the argument number: +1 for a positive number, −1 for a negative number, and 0 if the number is 0.

Example: Addition. Calculate the sum of 0.4_lbf and 11.9_dyne. First, enter the unit objects.

(UNITS) (NXT) FORCE
.4 LBF
11.9 DYN

```
2:              .4_lbf
1:            11.9_dyn
 N  DYN  GF  KIP  LBF  PDL
```

Add the unit objects. The unit conversion is done automatically.

(+)

```
1:    177940.76461_dyn
 N  DYN  GF  KIP  LBF  PDL
```

Example: Subtraction. Subtract 39_in from 4_ft.

(UNITS) LENG
4 FT
39 IN
(−)

```
1:                9_in
 M  CM  MM  YD  FT  IN
```

Example: Scalar Multiplication and Division. Multiply 12_mph by 10, then divide by 6.

(UNITS) SPEED
12 MPH
10 (×) 6 (÷)

```
1:              20_mph
 M/S  CM/S  FT/S  KPH  MPH  KNOT
```

Example: Unit Multiplication and Division. Multiply 50_ft by 45_ft, then divide by 3.2_d (days). First, multiply the two unit objects.

(UNITS) LENG
50 FT
45 FT
(×)

```
1:           2250_ft^2
 M  CM  MM  YD  FT  IN
```

Enter the third unit object and divide.

(←)(UNITS) TIME
3.2 D
(÷)

```
1:          703.125_ft^2/d
 YR | D | H | MIN | S | HZ
```

13

Example: Inversion. Find the reciprocal of `11.4_g*cm/s^2`.

11.4 (←)(UNITS) MASS G
(←)(UNITS) LENG CM
(←)(UNITS) TIME
(→) S (→) S
(1/x)

```
1: 8.77192982456E-2_s^
   2/(g*cm)
 YR | D | H | MIN | S | HZ
```

Example: Power. Raise `2_ft/s` to the sixth power. Find the square root of the result. Then find the cube root of that result.

Enter the unit object and raise the unit object to the sixth power.

2 (←)(UNITS) SPEED FT/S
6 (y^x)

```
1:             64_ft^6/s^6
 M/S | CM/S | FT/S | KPH | MPH | KNOT
```

Now find the square root of the result.

($\sqrt{x}$)

```
1:              8_ft^3/s^3
 M/S | CM/S | FT/S | KPH | MPH | KNOT
```

Find the cube root of the result.

3 (→)($\sqrt[x]{y}$)

```
1:                  2_ft/s
 M/S | CM/S | FT/S | KPH | MPH | KNOT
```

Example: Percentage. `4.2_cm^3` is what percent of `1_in^3`?

(←)(UNITS) VOL 1 IN^3
4.2 CM^3
(MTH) PARTS (NXT) %T

```
1:           25.6299725198
 MIN | MAX | MOD | % | %CH | %T
```

Example: Trigonometry. Calculate the sine of 45°.

(←)(UNITS) (NXT) (NXT) ANGL
45 ° (SIN)

```
1:           .707106781187
 ° | R | GRAD | ARCMI | ARCS | SR
```

Example: Algebraic Calculation. In algebraic syntax, calculate the tangent of 40 grad.

['] [TAN] 40 [→] [_] GRAD [EVAL]

```
1:             .726542528005
 °  | R  | GRAD | ARCMI | ARCS | SR
```


Working with Temperature Units

You work with temperature units the same ways you work with other units—*except* you must recognize and anticipate the difference between temperature *level* and temperature *difference*. For example, a temperature *level* of 0 °C means "freezing," but a temperature *difference* of 0 °C means "no change."

When °C or °F represents a temperature *level*, 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 temperature *difference*, then the temperature is a unit with no additive constant: 1 °C = 1 K, and 1 °F = 1 °R.

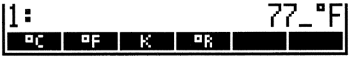
Converting Temperature Units

Conversions between the four temperature scales (K, °C, °F, and °R) involve additive constants as well as multiplicative factors. The additive constants are *included* in a conversion when the temperature units reflect actual temperature *levels*—they're *ignored* when the temperature units reflect temperature *differences*:

- **Pure temperature units (levels).** If both unit expressions consist of a single, unprefixed temperature unit with no exponent, the UNITS Catalog menu or CONVERT performs an *absolute* temperature scale conversion, which includes the additive constants.
- **Combined temperature units (differences).** If either unit expression includes a prefix, an exponent, or any unit other than a temperature unit, CONVERT performs a *relative* temperature unit conversion, which ignores the additive constants.

Example: Convert `25_°C` to `°F`.

(⬅) (UNITS) (NXT) TEMP
25 °C (⬅) °F



Example: Convert `20_°C/min` to `°F/s`. First, create the unit object `20_°C/min`.

(⬅) (UNITS) (NXT) TEMP
20 °C
(⬅) (UNITS) TIME (➦) MIN

Enter a unit object containing the new units.

(➦) (LAST MENU) 1 °F
(➦) (LAST MENU) (➦) S

Perform the conversion.

(➦) (UNITS) CONV

Calculating with Temperature Units

Temperature units are automatically converted and combined during calculations.

- **Pure temperature units (levels or differences).** For the +, −, =, <, >, ≤, ≥, ==, ≠, %, %CH, and %T functions, pure temperature units are interpreted as temperature *levels relative to absolute zero* for all temperature scales. Before making the calculation, the HP 48 converts any Celsius or Fahrenheit temperature to absolute temperatures. (This may give unexpected results if you actually intent the temperature units to mean temperature differences rather than temperature levels—for example, `0_°C 0_°C +` returns `273.15_°C`, twice as far from absolute zero as 0 °C.)

  For other functions, pure temperature units are interpreted as temperature *differences*—they're not converted before the calculation.

- **Combined temperature units (differences).** Temperature units with prefixes, exponents, or other units are interpreted as temperature *differences*—they're not converted before the calculation.

To enter a temperature difference for one of the six functions listed above, use absolute units (K or °R). For example, you can enter a difference of 2 °C as `2_K`.

To interpret a result from one of these functions as a temperature difference, convert the result to absolute units (K or °R). For example, a converted result of `2_K` means a temperature difference of 2 K or 2 °C.

Example: Determine if 12 °C is greater than 52 °F. (The > command interprets temperatures as levels.) The result shows the test is true (12 °C is 53.6 °F).

[⬅] [UNITS] [NXT] `TEMP`
12 `°C`
52 `°F`
[PRG] `TEST` [NXT] `>`

```
1:                1
 ==  ≠  <  >  ≤  ≥
```

Example: Calculate the temperature change from 29 °C to 17 °C. (Convert the calculated change to absolute units.)

[⬅] [UNITS] [NXT] `TEMP`
17 `°C` 29 `°C` [−]
[⬅] `K`

```
1:            -12_K
 °C  °F  K  °R
```

Example: Calculate the final temperature for an increase of 18 °F from the current temperature of 74 °F. (Enter the increase in absolute units.)

18 `°R` 74 `°F` [+]

```
1:            92_°F
 °C  °F  K  °R
```

Example: For a coefficient of linear expansion α of 20×10^{-6} 1/°C and a temperature change ΔT of 44 °C, calculate the fractional change of length given by $\alpha\,\Delta$T. (The $\times$ command interprets temperatures as differences.)

20 [EEX] [+/−] 6 [➡] `°C`
44 `°C` [×]

```
1:           .00088
 °C  °F  K  °R
```

Example: The ideal gas equation of state is $PV = nRT$, where P is the pressure exerted by the gas (in atmospheres), V is the volume of the gas (in liters), n is the amount of the gas (in moles), R is the ideal gas constant (0.082057 liter-atmospheres/kelvin-mole), and T is the temperature of the gas (in kelvins).

13

Assuming ideal gas behavior, calculate the pressure exerted by 0.305 mole of oxygen in 0.950 liter at 150 °C.

First, enter the temperature.

(←)(UNITS) (NXT) TEMP 150 °C	1: 150_°C °C | °F | K | °R | |

Convert the units to kelvins.

(←) K	1: 423.15_K °C | °F | K | °R | |

Multiply T (already in level 1) by n (0.305 mole).

(←)(UNITS) MASS (NXT)(NXT) .305 MOL (×)	1: 129.06075_K*mol U | MOL | | | |

Multiply nT by R.

.082057 (←)(UNITS) VOL (NXT) L (←)(UNITS)(NXT) PRESS ATM (←)(UNITS)(NXT) TEMP (→) K (←)(UNITS) MASS (NXT)(NXT)(→) MOL (×)	1: 10.5903379628_l*atm U | MOL | | | |

Divide by V (0.950 liter) to calculate P.

.95 (←)(UNITS) VOL (NXT) L (÷)	1: 11.1477241714_atm L | GALU | GALC | GAL | QT | PT

Convert the pressure (in atmospheres) to SI base units.

(→)(UNITS) UBASE	1: 1129543.15167_kg/(m *s^2) CONV | UBASE | UVAL | UFACT | →UNIT |

Note that in this example the temperature conversion from °C to K is executed *before* subsequent operations append additional units to the unit object. Otherwise, the Celsius temperature would have been treated as a difference, and the conversion to SI base units would have produced an incorrect result.

13

Creating User-Defined Units

If you use a unit that's not contained in the UNITS Catalog menu, you can create a *user-defined* unit that behaves just like a built-in unit.

To create a user-defined unit:

1. Enter a unit object using built-in or previously defined units that equals value of 1 new unit.
2. Store the unit object in a variable—the variable name is used as the name of the new unit.
3. Optional: Add a unit object with the user-defined unit to the CST menu—see below. The number part is ignored. (This procedure is explained under "Creating a Custom Menu" on page 15-1.)

You *can't* use the unit key in the VAR menu like unit keys in the UNITS menus—because VAR menu keys store and recall objects. However, if you add the user-defined unit to the CST menu, you can use the CST menu key to enter and convert your user-defined units—just like UNITS menu keys.

Example: Use the built-in unit d (day) to create the user-defined unit WEEK. (This example assumes there are no other entries in the CST menu list.)

Enter the unit object 7_d. Store the unit object in variable *WEEK*, then enter a list containing the unit object 1_WEEK.

[←] [UNITS] TIME 7 D
['] WEEK [STO]
[←] [{}] 1 [→] [_] [VAR] WEEK
[ENTER]

Store the list in the CST menu and display the menu. (This is explained under "Creating a Custom Menu" on page 15-1.)

[→] [MODES] MENU

`WEEK`

Now convert 14 days to weeks.

13 [←] [UNITS] TIME 14 D
[CST] [←] WEEK

`1: 2_WEEK`
`WEEK`

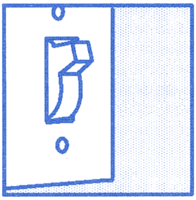
You can prefix a user-defined unit. However, conflicts between user-defined units (prefixed or otherwise) and built-in units are resolved in favor of the built-in unit.

Additional Commands for Unit Objects

Key	Programmable Command	Description
[→] [UNITS]:		
UVAL	UVAL	Returns the number part of the level 1 unit object to level 1.
→UNIT	→UNIT	Combines a number from level 2 with a unit object from level 1, ignoring the number part of the level 1 object, to form a unit object in level 1.

14

Binary Arithmetic



The HP 48 enables you to do binary arithmetic—operations that work with binary integers. You can think of a binary integer as a base 2 number—although it can be expressed in other number bases, too. When expressed in base 2, it consists of just 0's and 1's—each of which is a *bit*. Eight bits make up a *byte*.

On the HP 48, binary integer objects contain from 1 to 64 bits, depending on the current *wordsize*. You can enter and display binary integers in decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2). The *current base* determines which base is used to display binary integers on the stack.

The # delimiter precedes a binary integer. A d, h, o, or b following the binary integer indicates its base—for example, # 182d, # B6h, # 266o, or # 10110110b.

Setting the Wordsize

The wordsize is the number of bits used to represent binary integers. The wordsize can range from 1 through 64 bits—its default is 64 bits.

To set the wordsize:

1. Key in a number from 1 to 64.
2. Press (MTH) BASE STWS (the STWS command). (A fractional number is rounded to the nearest integer.)

To recall the current wordsize:

- Press (MTH) BASE RCWS (the RCWS command).

If you enter a binary integer that exceeds the current wordsize, the number is displayed with its most significant bits truncated—any bits over 64 are lost, and any bits from the current wordsize to 64 are
14 "hidden" (you can display them by increasing the wordsize). However, hidden bits are not used in calculations and are lost when you execute a command on a binary integer.

Also, the wordsize controls the results returned by arithmetic operations and other commands. If an argument exceeds the current wordsize, the excess most significant bits are dropped before the command is executed. If necessary, results are also truncated.

Setting the Current Base

Binary integers are displayed in decimal, hexadecimal, octal, or binary base. The default base is decimal.

To set the current base:

1. Press (MTH) BASE .
2. Press one of the following keys: HEX (hexadecimal), DEC (decimal), OCT (octal), or BIN (binary).

The ▪ in one of the menu labels identifies the current base.

HEX, DEC, OCT, and BIN are programmable. The settings for flags −11 and −12 correspond to the current base. (For more information on flags −11 and −12, see appendix E, "Listing of HP 48 System Flags.")

The choice of current base has no effect on the internal representation of binary integers.

Entering Binary Integers

To enter a binary integer:

1. Press (↱)(#).
2. Enter the value of the binary integer—valid characters depend on the base you're using.
3. Optional: To specify the base, type a base marker: d, h, o, or b. (Otherwise, the current base is used.)
4. Press (ENTER).

14

Example: Enter the address $24FF_{16}$ and display it in hexadecimal base.

(MTH) BASE HEX
(↱)(#) 24FF (ENTER)

```
1:                    # 24FFh
HEX■ DEC OCT BIN STWS RCWS
```

Now, display it in octal base.

OCT

```
1:                   # 22377o
HEX DEC OCT■ BIN STWS RCWS
```

Example: Enter 101101_2 while the current base is octal (from the previous example). (Press (α) (↰) B to type "b".)

(↱)(#) 101101b (ENTER)

```
1:                      # 55o
HEX DEC OCT■ BIN STWS RCWS
```

Calculating with Binary Integers

If an argument exceeds the current wordsize, the excess most significant bits are dropped before the command is executed. If necessary, results are also truncated. If a calculation produces a remainder, only the integer portion of the result is retained. The negative of a binary number is its two's complement (all bits inverted and 1 added).

To calculate with binary integers:

1. Enter the binary integer objects.
2. Execute the commands.

Example: Calculate $46AF_{16} - 33D_{16}$. First, switch to hexadecimal base and enter the two numbers.

(MTH) BASE HEX
(→)(#) 46AF (ENTER)
(→)(#) 33D (ENTER)

Execute the (−) command.

(−)

Example: Divide 64_{10} by 5_{10}. (The remainder of 4d is lost.)

(MTH) BASE DEC
(→)(#) 64
(→)(#) 5 (÷)

Additional Binary Integer Commands

The following table contains commands from the MTH BASE menu ((MTH) BASE) that are useful for manipulating binary integer objects. Unless otherwise stated, each example assumes the wordsize is set to 24.

Command/Description	Example	
	Input	Output
AND Logical bit-by-bit AND of two arguments.	2: # 1100b 1: # 1010b	1: # 1000b
ASR Arithmetic Shift Right. Performs 1 bit arithmetic right shift. The most significant bit is regenerated.	1:# 1100010b 1: # 800000h	1: # 110001b 1: # C00000h
B→R Binary to Real. Converts a binary integer to its real integer equivalent.	1: # 755o	1: 493
NOT Returns the one's complement of the argument. Each bit in the result is the complement of the corresponding bit in the argument.	1: # F0F0F0h	1: # F0F0Fh
OR Logical bit-by-bit OR of two arguments.	2: # 1100b 1: # 1010b	1: # 1110b
R→B Real to Binary. Converts a real integer to its binary integer equivalent.	1: 10	1: # 1010b

14

14

Command/Description	Example	
	Input	Output
RL Rotate Left. Binary integer rotates left one bit. (Example assumes wordsize=4.)	1: # 1100b	1: # 1001b
RLB Rotate Left Byte. Binary integer rotates left one byte.	1: # FFFFh	1: # FFFF00h
RR Rotate Right. Binary integer rotates right one bit. (Example assumes wordsize=4.)	1: # 1101b	1: # 1110b
RRB Rotate Right Byte. Binary integer rotates right one byte.	1: # A0B0C0h	1: # C0A0B0h
SL Shift Left. Binary integer shifts left one bit.	1: # 1101b	1: # 11010b
SLB Shift Left Byte. Binary integer shifts left one byte.	1: # A0B0h	1: # A0B000h
SR Shift Right. Binary integer shifts right one bit.	1: # 11011b	1: # 1101b
SRB Shift Right Byte. Binary integer shifts right one byte.	1: # A0B0C0h	1: # A0B0h
XOR Logical bit-by-bit exclusive OR of the arguments.	2: # 1100b 1: # 1010b	1: # 110b

15

Customizing the Calculator

The HP 48 provides several ways to customize its behavior. You can create *custom menus* containing the operations you want, you can set up your own functionality for the *user keyboard*, and you can control the calculator's *modes* using the MODES menu and by setting and clearing *flags*.

Using Custom (CST) Menus

A custom menu is a menu that you create. It can contain menu labels for operations, commands, and other objects that you create or group together for your own convenience.

Creating a Custom Menu

The CST menu is defined by the contents of a *reserved* variable named *CST*. So, the way to create a custom menu involves creating a variable *CST* that contains the objects you want in your menu.

To create and display the CST menu in the current directory:

1. Enter a list containing the objects you want in the menu. (The purposes of different object types are described below.)
2. Press (→)(MODES) MENU (the MENU command).

MENU stores the contents of the list in *CST* and displays the custom menu. Alternatively, you can create the CST menu by storing the custom-menu list in *CST* just like you would store a list in any variable—enter the list on the stack and press (VAR) (←) CST or

['] [⮫] [MODES] CST [STO]. (The MODES Customization menu always contains a menu label for *CST*.)

To display the CST menu:

- Press [CST].

15 Objects in the CST menu usually have the same functionality they do in built-in menus:

- **Names.** Names behave like VAR menu keys. Thus, if *ABC* is a variable name, ABC evaluates *ABC*, [⮫] ABC recalls its contents, and [⮪] ABC stores new contents in *ABC*. Also, the menu label for the name of a directory has a bar over the left corner of the label—pressing the menu key switches to that directory.
- **Units.** Unit objects act like UNITS Catalog entries. For example, they have their left-shifted conversion capability.
- **Strings.** String objects echo the contents of the string, like a typing aid.
- **Commands.** Almost all command names behave like normal command keys. For example, they observe the current entry mode.

You can include backup objects in the list defining a custom menu by tagging the name of the backup object with its port location. For example, if `:2:TOM` were included in the custom menu list, a menu label TOM would represent the backup object *TOM* in port 2.

If you want to create typing aids for certain commands that affect program flow (such as HALT, PROMPT, IF...THEN...END, and other program control structures), include them as string objects, not as command names.

Example: Create a custom menu containing the built-in command →TAG, the unit object `1_m^3`, a string to serve as a typing aid for `VOLUME`, and the variable name *CST*.

Enter the list of objects.

[⮪] [{ }] [PRG] OBJ →TAG
1 [⮫] [_] m [y^x] 3
[⮫] [" "] VOLUME [▶]
CST [ENTER]

```
1: { →TAG 1_m^3
     "VOLUME" CST }
OBJ→ EQ→ →ARR →LIST →STR →TAG
```

Create and display the CST menu.

(⮫) (MODES) MENU

```
→TAG | M^3 | VOLU | CST |  |
```

Convert 1075 cm^3 to m^3.

1075 (⮫) (_) cm (y^x) 3 (ENTER)
(⮨) M^3

```
1:             .001075_m^3
→TAG | M^3 | VOLU | CST |  |
```

Enter the string "VOLUME".

(⮫) (" ") VOLU (ENTER)

```
2:             .001075_m^3
1:                "VOLUME"
→TAG | M^3 | VOLU | CST |  |
```

Create a tagged object from the contents of levels 2 and 1.

→TAG

```
1: VOLUME: .001075_m^3
→TAG | M^3 | VOLU | CST |  |
```

Display the current contents of *CST*.

CST

```
2: VOLUME: .001075_m^3
1: { →TAG 1_m^3
   "VOLUME" CST }
→TAG | M^3 | VOLU | CST |  |
```

You can create a *CST* in each directory in memory, just as you can for other variables. This lets you have different custom menus in each directory.

Also, instead of storing the list of objects itself in *CST*, you can optionally store the name of another variable that contains the list. This gives you the ability to have in one directory several variables that contain different custom-menu lists. That way, you can easily switch the CST menu from one custom menu to another by simply storing a new name in *CST*.

Enhancing Custom Menus

You can enhance the CST menu by creating special menu labels and by specifying different actions for unshifted and shifted keys.

To create a special menu label for an object:

- Inside the CST list, replace the object by an embedded list of the form { "*label*" *object* }.

The default label for an object in the CST menu is the underlying name, command, unit, or typing aid—as many characters as fit in the space available.

Example: Storing `{ →TAG 1_m^3 { "VOL" "VOLUME" } { "CUST" CST } }` in *CST* gives the same CST menu operations as the previous example, but the labels are `→TAG`, `M^3`, `VOL`, and `CUST`.

15

To specify functionality for shifted keys:

- Inside the CST list, replace the object by an embedded list of objects: `{` *object*$_{\text{unshifted}}$ *object*$_{\text{left-shift}}$ *object*$_{\text{right-shift}}$ `}`. (You can omit the last one or two objects if you want.)

You must specify the unshifted action in order to have the shifted actions. In addition, you can combine the special-label enhancement and the shifted-functionality enhancement—see the following example.

Example: Suppose you want the CST menu key `VOL` to provide the following three actions:

- `VOL` evaluates a program that stores the value in level 1 in a variable named *VBOX*.
- ⬅ `VOL` evaluates a program that computes the product of levels 1, 2, and 3.
- ➡ `VOL` types `VOLUME`.

The following CST list provides the desired custom menu. The menu contains only one label: `VOL`. (See chapter 25 to learn about programs.)

```
{ { "VOL" { « 'VBOX' STO » « * * » "VOLUME" } } }
```

Creating a Temporary Menu

The TMENU command creates a temporary menu without overwriting the contents of the variable *CST*. Temporary menus are most useful in programming—they're covered under "Using Menus with Programs" on page 29-18.

Defining the User Keyboard

The HP 48 lets you assign alternate functionality to any key on the keyboard (including alpha and shifted keys), enabling you to customize the keyboard for your particular needs. Your customized keyboard is called the *user keyboard*, and it's active whenever the calculator is in *User mode*.

15

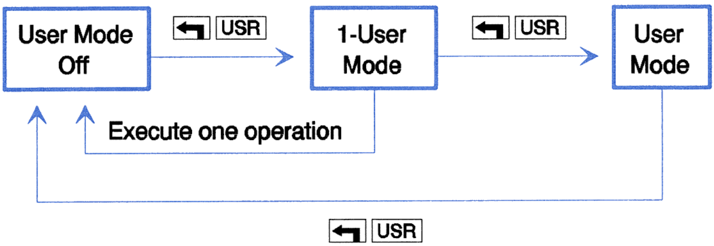
The commands for creating and changing the user keyboard are located in the MODES Customization menu (⮣ MODES).

Selecting User Modes

To activate User mode:

- To activate it for only one operation (1USR), press ⮢ USR. (It turns off after the operation.)
- To activate it for several operations (USER), press ⮢ USR ⮢ USR. (Press ⮢ USR a third time to turn off User mode.)

The ⮢ USR key is a three-way switch, as shown in the following illustration.



In 1-User mode (1USR annunciator), the user keyboard is active for one operation. In User mode (USER annunciator), the user keyboard remains active until you press ⮢ USR to turn it off.

If you set flag −61, ⮢ USR becomes a two-way switch between User mode on and User mode off. (See "Using System Flags" on page 15-12.)

To change the way ↰ USR operates:

- To make it a two-way switch, type 61 +/- ↱ MODES NXT SF.
- To make it a three-way switch, type 61 +/- ↱ MODES NXT CF.

15

Assigning and Unassigning User Keys

You can assign commands or other objects to any user keys (including shifted keys). The behaviors for different types of objects are the same as for custom menus—see “Creating a Custom Menu” on page 15-1.

To assign one user key:

1. Enter the object to be assigned to the key.
2. Enter the three-digit location number that specifies the key. (See the diagram below.)
3. Press ↱ MODES ASN (the ASN command).

Keyboard column#

↓

XX.X ← 0 or 1 = unshifted
2 = left-shifted
3 = right-shifted
4 = alpha
5 = alpha left-shifted
6 = alpha right-shifted

↑

Keyboard row#

If the object you're assigning is a built-in command, use the next procedure instead.

To assign several user keys:

1. Enter a list containing two key-assignment parameters for each key—the object to be assigned to the key followed by the three-digit key location number (see above).
2. Press ↱ MODES STOK (the STOKEYS command).

This is an example of a key-assignment list for STOKEYS:

`{ SINH 41 "3.14" 94.2 ABC 11.4 }`

You can use `'SKEY'` as an assignment object. It means the "standard" (unassigned) key definition.

When you press a user key, its assigned object is executed—or, if the key is unassigned, the standard operation is performed. (You can also disable keys, as described in the next topic.) 15

After you've assigned a user key, the assignment remains in effect until you reassign the key using ASN or STOKEYS, or until you *unassign* the key. An unassigned user key reverts to its standard definition—the same as for the standard keyboard.

Example: Assign the typing aid `VOLUME` to the [SIN] key (without otherwise affecting the user keyboard).

Press [→][" "] VOLUME [ENTER] to create the string object. Press 41 [→][MODES] `ASN` to assign the string to the [SIN] key (row 4, column 1, unshifted). Now, you can press [←][USR] [SIN] to type `VOLUME`.

Example: Assign the DUP2 command to [←][SWAP] (without otherwise affecting the user keyboard).

Press [←][{ }] [PRG] `STK` [NXT] `DUP2` 36.2 [ENTER] to create the key-assignment list. Then press [→][MODES] `STOK` to assign the [←][SWAP] key (row 3, column 6, left-shifted). Now, when the calculator's in 1-User or User mode, press [←][SWAP] to execute DUP2.

Example: Assign the standard definitions of the [NXT] and [←][USR] user keys.

Press [←][{ }] ['] SKEY [▶] 26 ['] SKEY [▶] 61.2 [ENTER] [→][MODES] `STOK`. (The key-assignment list is `{ 'SKEY' 26 'SKEY' 61.2 }`).

Alternatively, you can assign each key individually using ASN.

Example: Make the following user key assignments:

- Assign the variable *ABC* containing `{ A B C }` to [α] [A].
- Assign the program `« OBJ→ DROP »` to [→][_].
- Assign the command DROP2 to [←][DROP].
- Assign the string (typing aid) "HEIGHT" to [α] [←][h].

Create the variable *ABC* containing the list `{ A B C }` and display the VAR menu.

[←][{}] A [SPC] B [SPC] C [ENTER]
['] ABC [STO] [VAR]

```
ABC  WEEK  CST  BOXR  BOXS  COT
```

Enter the key-assignment list for the STOKEYS command.

[←][{}] ABC 11.4
[←][«»] [PRG] OBJ OBJ→
[←][DROP] [▶] [▶] 75.3
[PRG] STK [NXT] DROP2 55.2
[→][" "] HEIGHT [▶] 22.5 [ENTER]

```
1: { ABC 11.4 « OBJ→
   DROP » 75.3 DROP2
   55.2 "HEIGHT" 22.5
   }
DUP  DUP2  DUPN  DROP2  DRPN
```

Execute STOKEYS and activate the user keyboard.

[→][MODES] STOK
[←][USR] [←][USR]

```
ASN  STOK  RCLK  DELK  MENU  CST
```

Now, retrieve the list in *ABC* and use the program assigned to [→][_] to separate it into its components.

[α] [A]
[→][_]

```
3:                'A'
2:                'B'
1:                'C'
ASN  STOK  RCLK  DELK  MENU  CST
```

Execute DROP2, then put the string `"HEIGHT"` on the stack.

[←][DROP]
[→][" "] [α] [←][h] [ENTER]

```
2:                'A'
1:           "HEIGHT"
ASN  STOK  RCLK  DELK  MENU  CST
```

Press [←][USR] to turn off User mode.

To unassign previously assigned user keys:

- To unassign one user key, enter the three-digit key number, then press [→][MODES] DELK.
- To unassign several user keys, enter a list containing the three-digit key numbers, then press [→][MODES] DELK.
- To unassign all user keys, press 0 [→][MODES] DELK.

An unassigned user key reverts to its standard definition—the same as for the standard keyboard. If you use DELKEYS with argument 0 to unassign all user keys, all disabled keys are enabled (see the next topic).

Example: Unassign all user keys.

0 (→)(MODES) DELK

ASN	STOK	RCLK	DELK	MENU	CST

15

Disabling User Keys

You can *disable* user keys that are unassigned—so they do nothing. This lets you control the user keys that are active, including assigned keys and standard (unassigned) keys.

If you assign a disabled user key, it becomes enabled.

To disable all unassigned user keys:

- Enter 'S' and press (→)(MODES) DELK.

To enable and unassign disabled user keys:

- To enable one unassigned key, enter 'SKEY', enter the three-digit key number, then press (→)(MODES) ASN.
- To enable several unassigned keys, enter a list containing 'SKEY' and the three-digit key number for each key, then press (→)(MODES) STOK. (Include one 'SKEY' for each key.)
- To enable and unassign *all* user keys, press 0 (→)(MODES) DELK.

To enable and assign disabled user keys:

- To enable and assign one user key, enter the object to be assigned to the key, enter the three-digit key number, then press (→)(MODES) ASN.
- To enable all user keys and assign several keys, enter a list with S as the first object and followed by the assigned object and three-digit key number for each key assignment, then press (→)(MODES) STOK.

15

Recalling and Editing User Key Assignments

To recall the current user key assignments:

- Press (↱)(MODES) RCLK (the RCLKEYS command).

The RCLKEYS command returns to level 1 a list of all the current user key assignments—pairs of assignment objects and three-digit key numbers. If the first item in the list is the letter S, then unassigned user keys are currently enabled—otherwise, unassigned keys are currently disabled.

To edit the user key assignments:

1. Press (↱)(MODES) RCLK (the RCLKEYS command).
2. Press (↰)(EDIT) and edit the key-assignment list.
3. Press (↱)(MODES) STOK (the STOKEYS command) to activate the edited assignments.

Note

If you get stuck in User mode—probably with a "locked" keyboard—because you've reassigned or disabled the keys for canceling User mode, hold down the (ON) key and press the C key, then release the C key first.

Deleted user key assignments still take up from 2.5 to 15 bytes of memory each. You can free this memory by packing your user key assignments—press (↱)(MODES) RCLK 0 DELK STOK.

Setting Calculator Modes

You can use the MODES menu to set certain operating modes. To set these and other operating modes or conditions, you can set and clear system flags.

Using the MODES Menu

The multiple pages of the MODES menu ((←)(MODES)) contain operations that let you customize the way your calculator operates. When a menu label has a ■ in it, that mode is active. For example, SYM■ means Symbolic Results mode is active. (If you want to change one of these modes in a program, you must set or clear the appropriate flags—see the next topic.) 15

MODES Operations

Key	Description
(←)(MODES):	
SYM	Switches between symbolic (■ in label) and numeric evaluation.
BEEP	Switches between errors beeping (■ in label) and not beeping.
STK	Switches between saving (■ in label) and not saving the last stack. Affects the action of (←)(LAST STACK).
ARG	Switches between saving (■ in label) and not saving the last arguments. Affects the action of (→)(LAST ARG).
CMD	Switches between saving (■ in label) and not saving in memory the last command line. Affects the action of (←)(LAST CMD).
CNCT	Switches between drawing a continuous line to connect plotted points (CNC■) and plotting points only (CNCT).
ML	Switches between displaying a multiline level 1 as multiple lines (■ in label) and as a single line followed by an ellipsis.
CLK	Switches between displaying a clock (■ in label) and not displaying a clock.
FM,	Switches between decimal fraction mark and comma fraction mark (■ in label).

Using System Flags

The HP 48 provides a number of modes that also let you customize its operating environment. Most modes are controlled by *system flags*. The HP 48 has 64 system flags, numbered −1 through −64. Each flag can have two states—set (value of 1) or clear (value of 0). The system
15 flags and the modes they control are described in appendix E.

The commands for setting, clearing, and testing flags are in the MODES Customization menu ((↱)(MODES)). (They're duplicated in the PRG TEST menu.) They take flag numbers as arguments.

To use a flag command:

1. Enter the number of the flag (negative for a system flag).
2. Execute the command (see the table below).

Flag Commands

Key	Programmable Command	Description
(↱)(MODES) (pages 2 and 3) or (PRG) TEST (page 3):		
SF	SF	Sets the flag.
CF	CF	Clears the flag.
FS?	FS?	Returns true (1) if flag is set and false (0) if flag is clear.
FC?	FC?	Returns true (1) if flag is clear and false (0) if flag is set.
FS?C	FS?C	Tests flag (returns true (1) if set and false (0) if clear), then clears the flag.
FC?C	FC?C	Tests flag (returns true (1) if clear and false (0) if set), then clears the flag.

Example: Automatic Alpha Lock. Ordinarily, Alpha-entry mode is locked by pressing (α) twice in a row. You can choose instead to have a single press of (α) automatically activate alpha lock. To select Automatic Alpha Lock mode, set system flag −60: press 60 (+/−) (↱)(MODES) (NXT) SF .

Example: User-Mode Lock. Pressing (⬅)(USR) once normally puts your calculator in User mode for one keystroke—pressing it twice in a row locks User mode until you press it a third time. To have User mode "lock in" on the first press, set flag −61: press 61 (+/-) (➡)(MODES) (NXT) SF .

Example: Evaluating Symbolic Constants. Symbolic constants (e, i, π, MAXR, and MINR) normally retain their symbolic form when evaluated. If you want them to be automatically evaluated using their HP 48 numeric representations, set flag −2: press 2 (+/-) (➡)(MODES) (NXT) SF .

15

The previous examples show just a few of the ways you can use flags to customize the way your HP 48 operates. You can also use flags to affect the display, math operations, printing, plotting, time management, and various other operations. For the complete listing of all 64 system flags and what they affect, see appendix E.

Part 3

Power Tools

16

The EquationWriter Application

The EquationWriter application lets you enter and review algebraic expressions and equations in the form most familiar to you—the way they appear printed in books and journals, and the way you write them with pencil and paper.

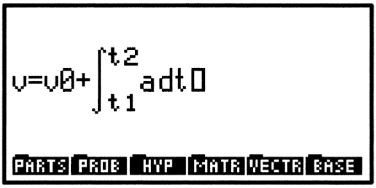
For example, here's an equation taken from a physics text:

$$v = v_0 + \int_{t_1}^{t_2} a\ dt$$

Here's how the equation would look on the stack:

```
'v=v0+∫(t1,t2,a,t)'
```

Now, here's the same equation keyed in using the EquationWriter application:

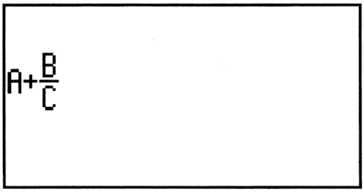


How the EquationWriter Application Is Organized

The EquationWriter application is a special environment where the keyboard is redefined and limited to special operations. Keys corresponding to algebraic functions enter the function name or graphical function symbol into the equation. For example, pressing ($\sqrt{x}$) draws a square root sign. You can display any menu—however, only those keys that correspond to algebraic functions are active. Like the function keys on the keyboard, the menu keys don't execute the corresponding function—they simply enter the function name into the equation.

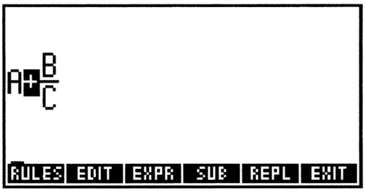
The EquationWriter application consists of three modes, each with its special purpose:

- **Entry mode.** For entering and editing equations.
- **Scrolling mode.** For viewing larger equations.
- **Selection mode.** For editing expressions within equations.



Scrolling Mode

Entry Mode



Selection Mode

Special keys on the keyboard are defined below.

Operations in the EquationWriter Application

Key	Description
▲	Starts a numerator.
▶ or ▼	Ends a subexpression. (⮫ ▶ or ⮫ ▼ ends all pending subexpressions.)
◀	Invokes *selection* mode, in which the Selection environment is active.
⬑ ( )	Enters (to start a parenthesized term. ▶ (or ▼) ends the parenthesized term.
SPC	Enters the current separator (, or ;) for multiple parenthetical arguments of functions and the terms of complex numbers.
EVAL	Exits the EquationWriter application and evaluates the equation.
ENTER	Returns the equation to the stack and exits the EquationWriter application.
ATTN	Exits the EquationWriter application without saving the equation.
⬑ GRAPH	Invokes *scrolling* mode. In scrolling mode, the menu keys are erased; if the equation is larger than the display, ▲ ▼ ◀ ▶ scroll the display window over the equation in the indicated direction. Press ⬑ GRAPH again (or ATTN) to return to the previous mode.
⬑ EDIT	Returns the equation to the command line for editing. (See "Editing Equations" on page 16-16.)
STO	Returns the equation to the stack as a *graphics object*. (See "The Structure of the PLOT Application" on page 18-2 and "Working with Graphics Objects on the Stack" on page 19-26 for discussions of graphics objects.)

Operations in the EquationWriter Application (continued)

Key	Description
(↱)(CLR)	Erases the display without leaving the EquationWriter application.
(↱)(RCL)	Inserts the level 1 object into the equation at the cursor position. (See "Editing Equations" on page 16-16.)
(↰)({ })	Turns *implicit parentheses* mode off. Press (↰)({ }) again to turn implicit parentheses mode back on. (See "Controlling Implicit Parentheses" on page 16-11.)
(↱)(" ")	Returns the equation to the stack as a string.

16

Constructing Equations

To start the EquationWriter application:

- Press (↰)(EQUATION).

After you start the EquationWriter application, you can enter an equation or expression (or unit object) using the operations available in this environment. See "Entering Equations" below.

To view a large equation or unit object:

1. Press (↰)(GRAPH) to activate scrolling mode.
2. Press (◀) (▶) (▲) (▼) to move the viewing "window."
3. Press (↰)(GRAPH) to return to the previous mode.

To exit the EquationWriter application:

- To put the equation on the stack and exit, press (ENTER).
- To discard the current equation and exit, press (ATTN).

Entering Equations

While you're entering an equation, you don't have to wait for the HP 48 to display the result of each key—you can keep typing because the HP 48 can remember up to 15 keystrokes.

To enter numbers and names:

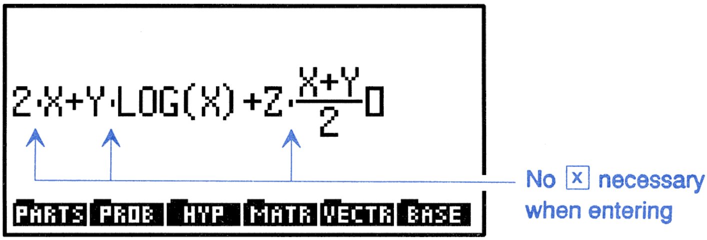
- Key them in exactly the same way you key them into the command line. The menu keys in the VAR menu act as typing aids for variable names.

To include addition, subtraction, and multiplication:

- To enter +, −, and ·, press [+], [−], and [×].
- To do *implied* multiplication, don't press [×]. (See below.)

You can do implied multiplication (without pressing [×]) in some situations—a multiply sign (·) is automatically inserted between:

- A number followed by an alpha character, a parenthesis, or a prefix function (a function whose argument(s) appear after its name)—for example, when you press 6 [SIN].
- An alpha character and a prefix function—for example, when you press A [←][x^2].
- A right parenthesis followed by a left parenthesis.
- A number or alpha character and the divide bar, square-root symbol, or xth root term—for example, when you press B [▲].

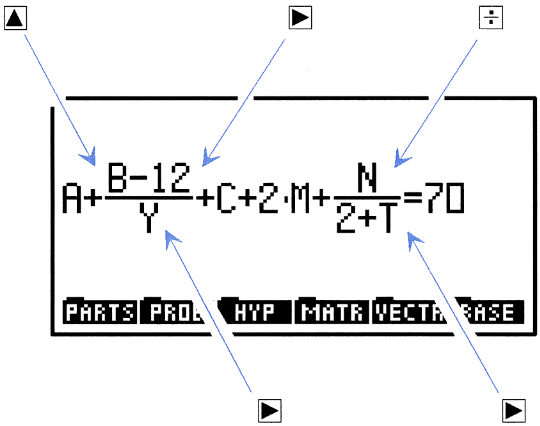


To include division and fractions:

1. Press [▲] to start the numerator.
2. Press [►] to end the numerator ([▼] works too).
3. Press [►] to end the denominator.

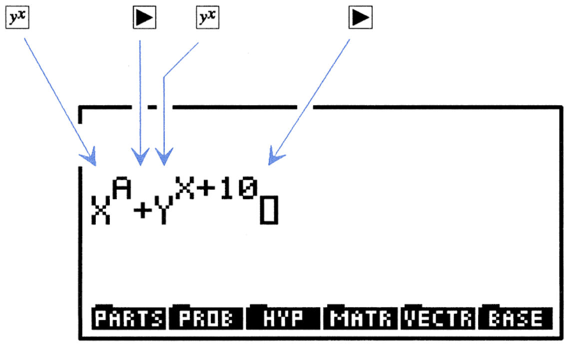
Here's an alternate way for fractions whose numerator consists of either *one* term or a sequence of terms with operators of precedence greater than or equal to / (divide):

1. Type the numerator (without pressing ▲).
2. Press ÷ to start the denominator.
3. Press ▶ to end the denominator (▼ works too).



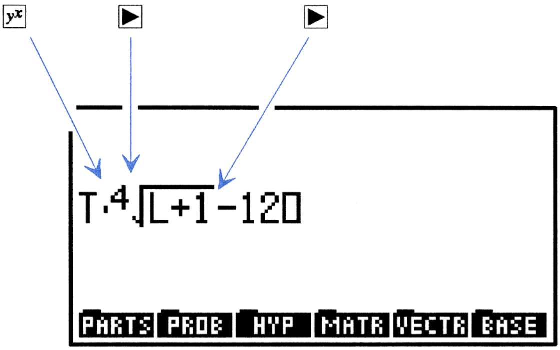
To include exponents:

1. Press y^x to start the exponent.
2. Press ▶ to end the exponent (▼ works too).



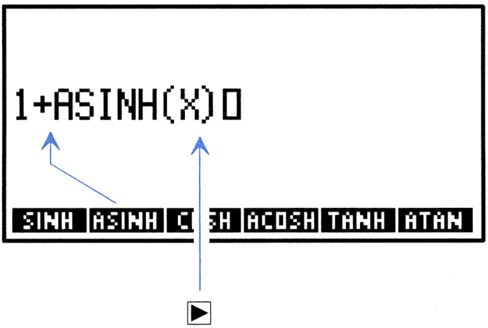
To include roots:

- To include square root, press [√x] to draw the √ symbol and start the term, then press [▶] to end the square root term.
- To include xth root, press [↱][$\sqrt[x]{y}$] to start the x term outside the √ symbol, press [▶] to draw the √ symbol and start the y term inside the √ symbol, then press [▶] to end the xth root term.



To include functions with parenthetical arguments:

1. Press the function key—or type the name and press [↰][( )].
2. Press [▶] to end the argument and display).



To include parenthesized terms:

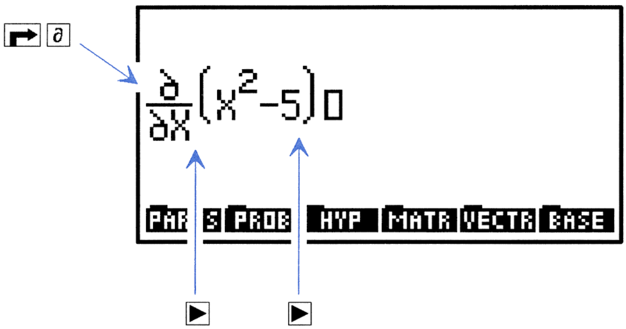
1. Press [↰][( )] to display the (.
2. Press [▶] to end the term and display).

To include powers of 10:

1. Press (EEX) to display E.
2. If the power is negative, press (+/–) to display -.
3. Key in the digits of the power.
4. Press any function key to end the power.

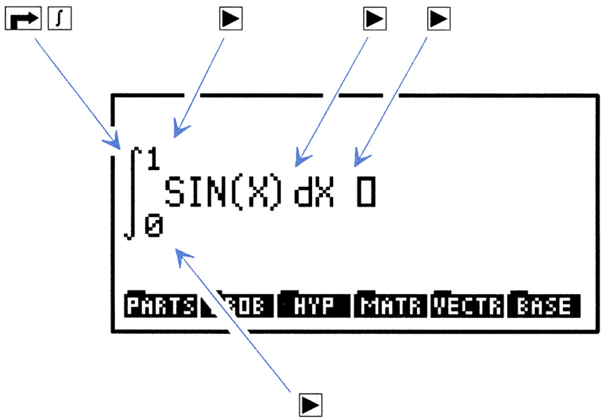
To include derivatives:

1. Press (↱)(∂) to display $\frac{\partial}{\partial}$.
2. Key in the variable of differentiation, then press (▶) to end the denominator and display ‹.
3. Key in the expression.
4. Press (▶) to end the expression and display ›.



To include integrals:

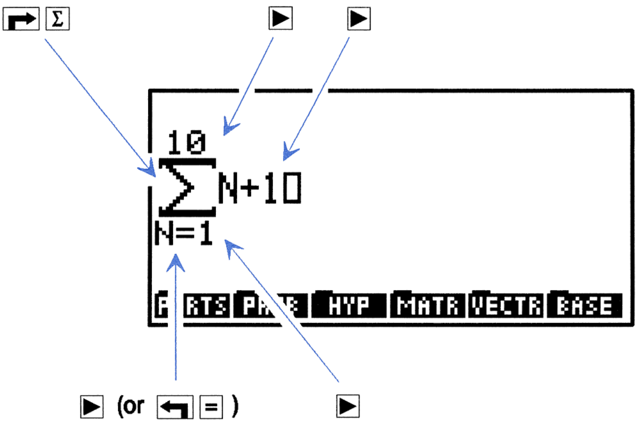
1. Press (↱)(∫) to display the ∫ symbol with the cursor positioned at the lower limit.
2. Key in the lower limit and press (▶).
3. Key in the upper limit and press (▶).
4. Key in the integrand and press (▶) to display d.
5. Key in the variable of integration.
6. Press (▶) to complete the integral.



16

To include "sigma" summations:

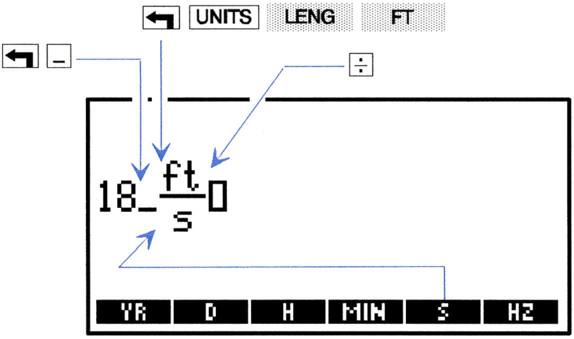
1. Press ⮫ Σ to display the Σ symbol. The cursor is positioned below.
2. Key in the summation index.
3. Press ▶ (or ⮪ =) to key in the = sign.
4. Key in the initial value of the index and press ▶.
5. Key in the final value of the index and press ▶.
6. Key in the summand.
7. Press ▶ to end the summation.



To include units:

1. Key in the number part of the unit object.
2. Press [↱] [_] to start the unit expression part of the unit object.
3. Key in the unit expression.
4. Press [▶] to end the expression.

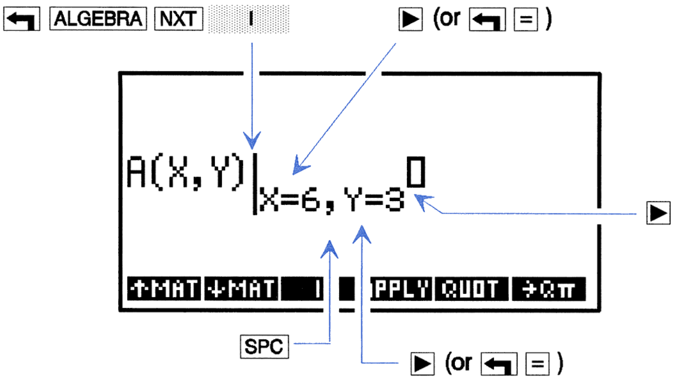
You can build unit objects (described in chapter 13) in the EquationWriter application. For combination units, press [×] or [÷] to separate each individual unit in the unit expression. You can key in unit names in one keystroke by pressing the corresponding menu key in the UNITS Catalog menu.



To include | (where) functions:

1. Key in a parenthetic expression with symbolic arguments.
2. Press [↰] [ALGEBRA] [NXT] | to display |. The cursor is positioned at the bottom right of the symbol.
3. Key in the defining equation for each argument, pressing [▶] or [↰] [=] to key in =, and [SPC] to key in the separator between each equation.
4. Press [▶] to end the function.

The | (where) function substitutes values for names in expressions. It's described under "Using the | (Where) Function" on page 22-25.



Controlling Implicit Parentheses

Implicit parentheses are turned on whenever you start the EquationWriter application. This means the arguments for [÷], [√x], and [y^x] are normally enclosed in "invisible" parentheses, so that only [▶] (or [▼]) ends the argument.

If you turn off implicit parentheses, the argument ends when you enter the next function—pressing [▶] doesn't end the argument.

To turn implicit parentheses on or off:

- Press [←][{ }]. A message showing the current state is displayed briefly.

Disabling implicit parentheses is convenient for entering polynomials, for example, where exponents are completed when you enter the function that starts the next term.

Leaving and then restarting the EquationWriter application turns implicit parentheses on. If you turn off implicit parentheses after keying in [÷], [√x], or [y^x], but before supplying the argument, implicit parentheses is *not* applied to those arguments.

The second example below demonstrates turning off implicit parentheses.

EquationWriter Examples

At the end of each example, instead of pressing [ENTER] to put the equation on the stack, you can press [↱][CLR] to clear the display for the next example. (If you press [↱][CLR], ignore the [↰][EQUATION] instruction at the start of each new example.)

If you make a mistake while you're keying in an equation, press [←] to backspace to the error—or press [↱][CLR] and start again. Note that the HP 48 may take several seconds to redisplay the equation after you've pressed [←] several times. In "Command-Line Editing" on page 16-17, you'll learn how to edit an equation in the command line.

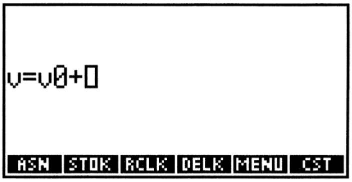
16

Example: Key in the equation

$$v = v_0 + \int_{t_1}^{t_2} a \, dt$$

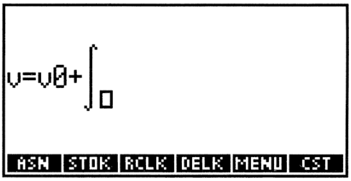
Select the EquationWriter application and key in the equation up to the $\int$ sign. (After you press [↰][EQUATION], you can lock lowercase alpha for alpha entry by pressing [α][↰][α].)

[↰][EQUATION]
v [↰][=] v0 [+]



Key in the integral sign.

[↱][∫]

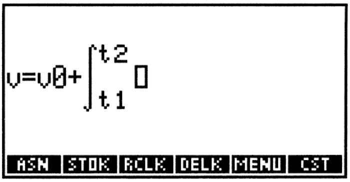


Key in the lower limit and move the cursor to the upper limit.

t1 [▶]

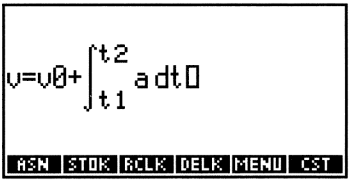
Key in the upper limit and move the cursor to the beginning of the integrand.

t2 (▶)



Key in the integrand and the variable of integration.

a (▶) t



Put the equation on the stack.

(ENTER)

Example: Key in the expression $X^3 + 2X^2 - \frac{1}{X}$, first with implicit parentheses and then without.

(◀)(EQUATION) X (y^x) 3 (▶) (+)
2X (y^x) 2 (▶) (−)
1 (÷) X

Clear the display and turn off implicit parentheses.

(▶)(CLR)
(◀)({})

Key in the expression again.

X [y^x] 3 [+] 2X [y^x] 2 [−] 1 [÷] X

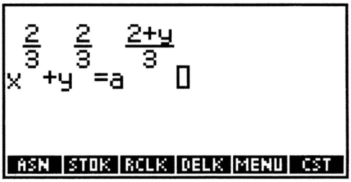
Press [←][{ }] [ENTER] to turn implicit parentheses on and put hte expression on the stack.

Example: Key in the equation

$$x^{\frac{2}{3}} + y^{\frac{2}{3}} = a^{\frac{2+y}{3}}$$

Key in the equation. (After you press [←][EQUATION], you can activate lowercase alpha lock by pressing [α][←][α].)

[←][EQUATION]
[α][←][α]
x [y^x] 2 [÷] 3 [▶] [▶] [+]
y [y^x] 2 [÷] 3 [▶] [▶] [←][=]
a [y^x] [▲] 2 [+] y [▶] 3
[▶][▶]

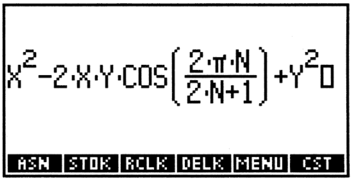


Press [ENTER] to put the equation on the stack.

Example: Key in the expression

$$X^2 - 2XY\cos\frac{2\pi N}{2N+1} + Y^2$$

[←][EQUATION]
X [y^x] 2 [▶] [−]
2X [×] Y [COS]
2 [←][π] [×] N [÷]
2N [+] 1 [▶] [▶]
[+] Y [y^x] 2 [▶]

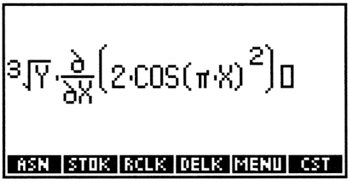


Press [ENTER] to put the expression on the stack.

Example: Key in the expression

$$\sqrt[3]{Y}\,\frac{d}{dX}2\cos^2(\pi X)$$

↰ EQUATION
↱ $\sqrt[x]{y}$ 3 ▶ Y ▶
↱ ∂ X ▶
2 COS ↰ π × X ▶
y^x 2 ▶ ▶



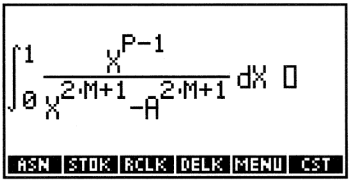
Press ENTER to put the expression on the stack.

16

Example: Key in the expression

$$\int_0^1 \frac{X^{P-1}}{X^{2M+1} - A^{2M+1}}\,dx$$

↰ EQUATION
↱ ∫ 0 ▶ 1 ▶
X y^x P − 1 ▶ ÷
X y^x 2M + 1 ▶
− A y^x 2M + 1 ▶ ▶
▶ X ▶

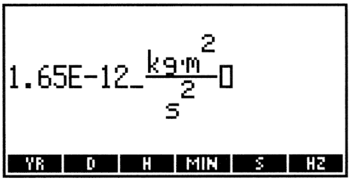


Press ENTER to put the expression on the stack.

Example: Key in the expression

$$1.65 \times 10^{-12}\frac{\text{kg}\cdot\text{m}^2}{\text{s}^2}$$

↰ EQUATION
1.65 EEX +/− 12 ↱ _
↰ UNITS MASS KG ×
↰ UNITS AREA M^2 ▶ ÷
↰ UNITS TIME S y^x 2
▶ ▶



Press ENTER to put the expression on the stack.

Editing Equations

You have several options for editing equations in the EquationWriter application:

- Backspace editing.
- Command-line editing.
- Inserting an object from the stack into the equation.
- Replacing a subexpression with an algebraic from the stack.

Backspace Editing

16 If you make a mistake while entering an expression in the EquationWriter application, you can erase characters back to the error and fix it.

To edit by backspacing:

- Press [◆] until you delete the error.
- Complete the expression correctly.

Note, however, that backspacing may be slow in certain situations. Backspacing is usually appropriate for correcting a mistyped character or digit—for extensive editing, use command-line editing, described in the next topic.

Example: Key in the expression $\sin(x + \sqrt{y + 180} + z)$.

Select the EquationWriter application and start the expression. "Accidentally" key in `170` instead of `180`.

[↰] [EQUATION]
[SIN] x [+]
[√x] y [+] 170

Backspace over 70, key in the correct number, and finish the subexpression.

(◀) (◀) 80 (▶)

Finish keying in the expression.

(+) z (▶)

Press (ENTER) to put the expression on the stack.

Command-Line Editing

You can edit all or part of an equation in the command line and then return the edited version to the EquationWriter application.

To edit the full equation:

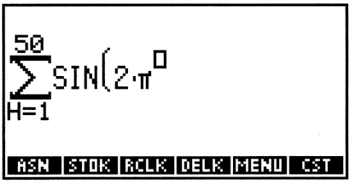
1. If the equation ends in an incomplete subexpression, complete it.
2. Press (↰)(EDIT).
3. Edit the equation in the command line.
4. Press (ENTER) to save the changes (or press (ATTN) to discard them) and return to the EquationWriter application.

Example: Key in the expression

$$\sum_{i=1}^{50} \sin(2\pi^i)$$

Select the EquationWriter application and key in the expression. "Accidentally" specify the series index as H instead of I.

(⬑) (EQUATION)
(⬏) (Σ)
H (▶) 1 (▶) 50 (▶)
(SIN) 2 (⬑) (π) (y^x)



Suppose that you now realize that you need to change the H to I. Try using command-line editing.

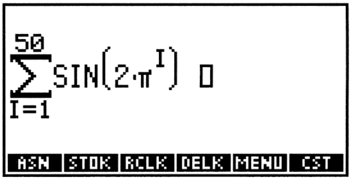
(⬑) (EDIT)

The EquationWriter application briefly displays the message `Incomplete Subexpression` and leaves the cursor at the end of the equation. Complete the expression, then start command-line editing.

I (▶) (⬑) (EDIT)

Change the H to I and return the expression to the EquationWriter application. (The HP 48 takes a few seconds to return the expression to the EquationWriter application.)

(▲) (▶) (DEL) I
(ENTER)



Press (ENTER) again to put the expression on the stack.

To edit a subexpression of an equation:

1. If the equation ends in an incomplete subexpression, complete it.
2. Press (◀) to activate the Selection environment.

3. Press (▲) (▼) (◀) (▶) to move the selection cursor to the *top-level function* for the subexpression you want to edit. (See below.)
4. Optional: Press EXPR at any time to show the associated subexpression—a highlight turns on or off.
5. Press EDIT to put the current subexpression in the command line.
6. Edit the subexpression in the command line.
7. Press (ENTER) to enter the revised subexpression into the equation (or press (ATTN) to discard it).
8. Press EXIT to leave the Selection environment. (If EXIT isn't displayed, press (−) to return to the Selection menu.)

The *Selection environment* is a special part of the EquationWriter application used to specify a subexpression in the equation.

A *subexpression* consists of a function and its arguments. The function that defines a subexpression is called the *top-level* function for that subexpression. For example, in the expression 'A+B*C/D', the top level function for the subexpression 'B*C' is *, the top-level function for 'B*C/D' is /, and the top level function for 'A+B*C/D' is +. (You can actually specify an individual object, a name for example, as the subexpression.)

You can also use the Selection environment to specify a subexpression to rearrange using the Rules transformations—see "Using the Rules Transformations" on page 22-11.

Example: Key in the expression

$$\tan\frac{4}{x}\int_0^1 x^y\,dx$$

Select the EquationWriter application and start the expression. In the argument for TAN, "accidentally" press (×) instead of (÷).

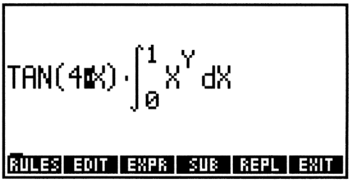
(◀) (EQUATION)
(TAN) 4 (×) X (▶)
(▶) (∫) 0 (▶) 1 (▶)

At this point you realize your mistake. However, you must enter the remaining arguments for the integral subexpression before activating the Selection environment.

X [y^x] Y [▶] [▶] X

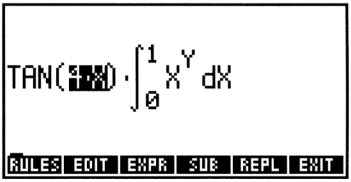
Now activate the Selection environment. Then move the cursor back to the unintended ·.

16 [◀]
[◀] (as required)



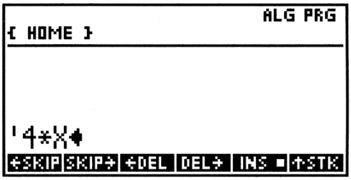
Optional: Highlight the corresponding subexpression for ·.

EXPR



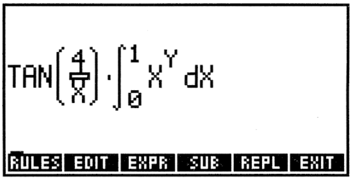
Return the subexpression to the command line for editing.

EDIT



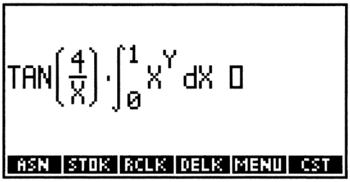
Replace the * with / and enter the change. (The HP 48 takes a few seconds to return the expression to the EquationWriter application.)

[◀] [←] [÷]
[ENTER] [−]



Leave the Selection environment. After several seconds, the normal cursor reappears at the end of the equation and the last menu is redisplayed.

EXIT



Press ENTER to put the expression on the stack.

Inserting an Object from the Stack

The EquationWriter application lets you insert an object from the stack into an equation you're entering. The object can be a name, a real number, a complex number, an algebraic, or a string.

To insert an object from level 1 into an equation:

- Press ⮫ RCL.

The object is deleted from the stack and inserted at the cursor position. The delimiters for names, algebraics, and strings are automatically removed.

Example: Enter the expression

$$\int_0^{10} x^2 - y \, dx + \frac{x^2 - y}{2}$$

Enter the expression 'X^2-Y' in the command line and duplicate it.

' X y^x 2 − Y ENTER ENTER

Select the EquationWriter application and key in the integral sign and limits of integration.

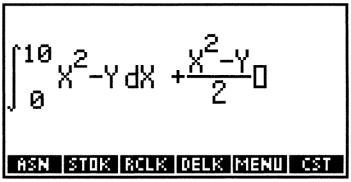
⮪ EQUATION
⮫ ∫ 0 ▶ 10 ▶

Insert the integrand into the expression.

[↱] [RCL]

Complete the subexpression. Then key in the remainder of the expression, inserting the second term from the stack.

[▶] X [▶]
[+]
[▲] [↱] [RCL] [▶] 2 [▶]



Press [ENTER] to put the expression on the stack.

Replacing a Subexpression with an Algebraic Object

The EquationWriter application lets you replace a subexpression or individual object in an equation. It's replaced by an algebraic object taken from the stack.

To replace a subexpression with an algebraic from level 1:

1. If the equation ends in an incomplete subexpression, complete it.
2. Press [◀] to activate the Selection environment.
3. Press [▲] [▼] [◀] [▶] to move the selection cursor to the *top-level function* for the subexpression you want to replace. (See "Command-Line Editing" on page 16-17.)
4. Optional: Press EXPR at any time to show the associated subexpression—a highlight turns on or off.
5. Press REPL .
6. Press EXIT to leave the Selection environment.

The algebraic is deleted from the stack.

Viewing and Editing Objects with the EquationWriter Application

You can use the EquationWriter environment to view and edit an algebraic or unit object in its EquationWriter form.

To view an existing algebraic or unit object with the EquationWriter application:

1. View the object:
   - If the object is in level 1, press (▼).
   - If the object is stored in a variable, put the variable name in level 1 and press (↱)(▼).
2. Press (ATTN) (ATTN) to return to the stack.

16

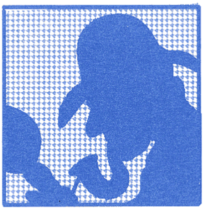
Note that, depending on the length and complexity of the algebraic or unit object, the HP 48 may take several seconds to display it in the EquationWriter application.

To edit an object you're viewing with the EquationWriter application:

1. Use any of the three EquationWriter modes—scrolling mode, selection mode, and entry mode. See "How the EquationWriter Application Is Organized" on page 16-2.
2. Press (ENTER) to save the changes (or press (ATTN) to discard them) and return to the stack.

17

The HP Solve Application



The HP Solve application lets you numerically solve real-valued equations containing any number of variables. It's a convenient alternative to symbolic math and programming when you want real-valued numeric results.

17

To solve an equation for numeric answers by hand, you might use the following general procedure:

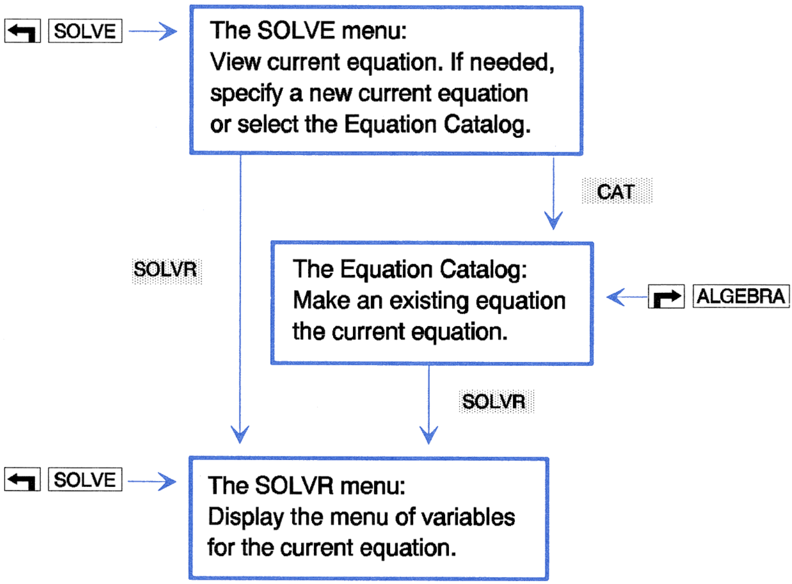
1. Write down the equation you want to solve.
2. If possible, manipulate the equation to solve for the unknown variable.
3. Substitute known values for the given variables.
4. Calculate the value of the unknown variable.

When you use the HP Solve application, you follow a similar procedure—*except* you don't need to do step 2, and that simplifies the process. And you can repeat steps 3 and 4 as often as you like, changing the values of one or more variables and solving for any variable.

The Structure of the HP Solve Application

The HP Solve application consists of two menus (the SOLVE menu and the SOLVR menu) and the reserved variable *EQ* containing the *current equation*—the equation you want to solve.

- The SOLVE menu lets you view the current equation or specify a new current equation. The SOLVE menu also lets you access the *Equation Catalog*, so you can select and manage existing equations.
- The SOLVR menu displays the variables for the current equation, letting you store, solve for, and review the numeric value of each variable in the equation.



Using Equations, Expressions, and Programs

The HP Solve application can solve for the numeric value of a variable in an equation, expression, or program:

- **Equation.** An equation is an algebraic object containing = (for example, `'A+B=C'`). A solution is a value of the unknown variable that causes both sides to have the same numeric value.
- **Expression.** An expression is an algebraic object not containing = (for example, `'A+B+C'`). A solution is a *root* of the expression—a value of the unknown variable for which the expression has a value of 0.
- **Program.** A program to be solved must return one real number. A solution is a value of the unknown variable for which the program returns 0.

17

Throughout this chapter, the term "equation" refers to all objects used to create SOLVR menus—equations, expressions, programs, and lists of equations, expressions, and programs.

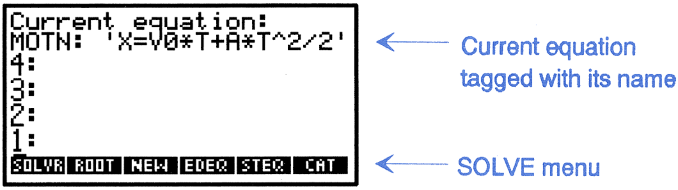
Specifying the Current Equation

The *current equation* is the equation you last solved or plotted. It's stored in reserved variable *EQ*. You change the current equation each time you solve or plot a different equation. (To *solve* the current equation, see "Solving the Current Equation" on page 17-12.)

To check the current equation:

- Press [←] [SOLVE].

A two-line status message gives the current equation and its name—or, if there's no current equation, it gives instructions for entering a new equation. In addition, the SOLVE menu is displayed.



Entering a New Current Equation

You can use either NEW or STEQ to enter a new current equation. NEW helps you enter and name a new current equation
17 by displaying an instructive message. STEQ is useful if you want to store an equation in *EQ* without naming it. Named equations are stored in variables so you can use them again later—unnamed equations are lost when you change the current equation.

To enter and name a new current equation:

1. Enter the equation in level 1. You can type it in the command line or use the EquationWriter application.
2. Press (⬅)(SOLVE) NEW.
3. Without pressing (α), key in a name for the equation and press (ENTER).

The equation is stored in a variable with the name you entered—the variable name is stored in *EQ*. (If you press (ENTER) without keying in a name, the equation itself is stored directly in *EQ*.)

If the "equation" is a program or list, NEW automatically adds .EQ to the variable name (or ,EQ if the fraction mark is ","). This identifies the variable as containing an object for solving or plotting.

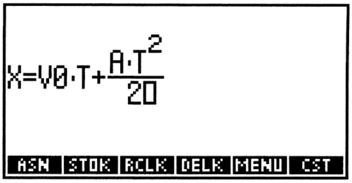
Example: Use NEW to enter and name the following equation for the motion of an accelerating body:

$$x = v_0 t + \frac{at^2}{2}$$

(This equation assumes that variables X, $V0$, T, and A don't exist in the current directory.)

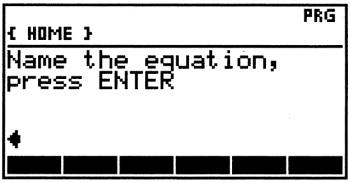
Key in the equation using the EquationWriter application.

(⮫)(CLR)
(⮨)(EQUATION)
X (⮨)(=) V0 (×) T
(+)(▲) A (×) T (y^x) 2 (▶)
(▶) 2



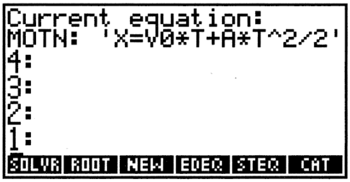
Store the equation as the current equation.

(ENTER)
(⮨)(SOLVE) NEW



In response to the prompt, name the equation MOTN. Don't press (α) because NEW automatically locks Alpha-entry mode.

MOTN (ENTER)



This equation is solved in the example on page 17-13.

To enter a current equation without naming it:

1. Specify the equation in level 1:
   - For a new equation, enter it into level 1.
   - For an equation stored in a variable, enter the variable name. (Press ('), then press the variable's menu key or type its name.)
2. Press (⮨)(SOLVE) STEQ (the STEQ command).

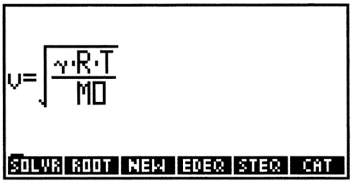
The STEQ command stores the equation or name in *EQ*. Note that an unnamed equation in *EQ* is lost the next time you store a new equation in *EQ*.

Example: Enter the equation below for the velocity of sound in a gas, then store it in variable *VSOUND*. Use STEQ to make it the current equation.

$$v = \sqrt{\frac{\gamma RT}{M}}$$

Select the EquationWriter application and key in the equation. (To key in γ, press (α) (↱) G.)

(↰)(EQUATION)
v (↰)(=) (√x)
γ (×) R (×) T (÷) M

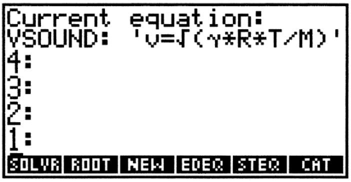


Store the equation in *VSOUND*.

(ENTER)
(') VSOUND (STO)

Use STEQ to make *VSOUND* the current equation.

(VAR) (') VSOU
(↰)(SOLVE) STEQ



Reusing an Existing Equation

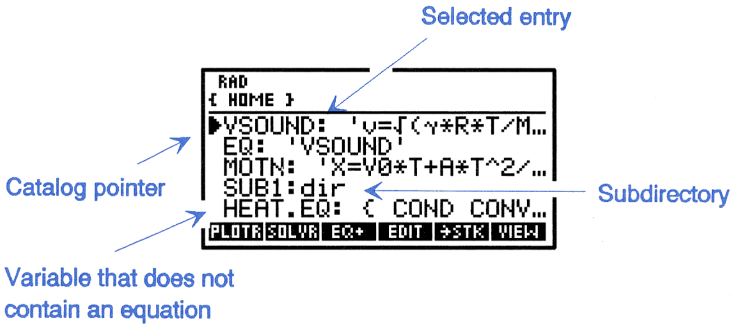
The Equation Catalog shows a listing of all named equations in the current directory. It's a special environment tailored to manage existing equations. The stack display is replaced by the equation listing, and the keyboard is redefined to execute special operations. These operations let you select the current equation and combine, edit, reorder, and purge existing equations. The Equation Catalog lists all named algebraics, plus all variable names ending in *EQ* and all directories in the current directory.

To get the Equation Catalog:

- Press (⬅)(SOLVE) CAT .

  or
- Press (⬅)(PLOT) CAT .

  or
- Press (➡)(ALGEBRA).



17

To work with an equation in the Equation Catalog:

1. Press (▲) and (▼) to move the pointer to the desired entry in the list.
2. Do the operation:
   - To make the equation the current equation and start solving the equation using the HP Solve application, press SOLVR.
   - To view the equation, press *and hold* VIEW —release the key to quit viewing.

To exit the Equation Catalog:

- To exit, update the current equation, and start solving the equation using the HP Solve application, press SOLVR.
- To exit without updating the current equation, press (ATTN).

The following table gives a complete list of operations in the Equation Catalog environment. The operations work on the selected entry. The Equation Catalog is used by the HP Solve application and by the Plot application (described in chapter 18).

Operations in the Equation Catalog

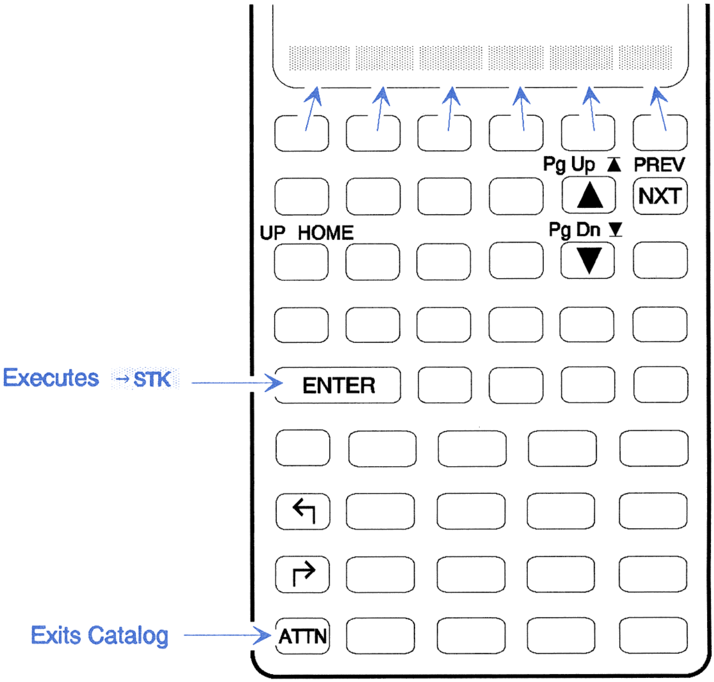
Key	Description
PLOTR	Makes the selected entry the current equation and displays the PLOTR menu.
SOLVR	Makes the selected entry the current equation and displays its menu of variables.
EQ+	Creates or adds to a list of equations. (See "Solving Two or More Equations" on page 17-27.) ([←] EQ+ removes the last entry from the list.)
EDIT	Places the selected entry in the command line for editing. Press [ENTER] to save the changes—or press [ATTN] to discard the changes.
→STK	Copies the selected entry to the stack.
VIEW	Clears the display and shows *only* the selected entry, without its name, until the key is released. If the selected entry is a directory, VIEW switches to that directory.
ORDER	Makes the selected entry the first entry in the catalog. If you create a list of n equations with EQ+, ORDER makes those equations the first n entries in the catalog.
PURG	Purges the selected entry from the catalog (and from the current directory).
FAST	Enabling FAST■ shows the names in the catalog (and in status messages) without their contents. (Enabling FAST■ sets flag −59. The converse is also true.) FAST■ is useful if the catalog contains many long equations, since such equations are slow to display.
[▲]	Moves the catalog pointer up one level. When prefixed with [←], moves the catalog pointer up one page ([←][PgUp] in the following keyboard illustration); when prefixed with [→], moves the catalog pointer to the top of the catalog ([→][⤒] in the following keyboard illustration).

17

Operations in the Equation Catalog (continued)

Key	Description
▼	Moves the catalog pointer down one level. When prefixed with ⬅, moves the catalog pointer down one page (⬅ PgDn in the following keyboard illustration); when prefixed with ➡, moves the catalog pointer to the bottom of the catalog (➡ ▼ in the following keyboard illustration).
ATTN	Exits the Equation Catalog.
ENTER	Executes →STK (copies the selected *equation* to the stack). If the selected entry is a *directory*, switches to the Equation Catalog in that directory.
⬅ UP	Switches to the Equation Catalog of the parent directory.
➡ HOME	Switches to the Equation Catalog of the *HOME* directory.

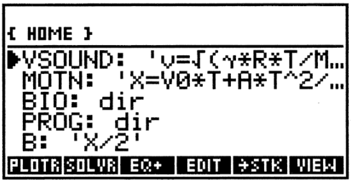
17



Example: Use the Equation Catalog to select *MOTN* (from the example on page 17-4) as the current equation and start solving it (by displaying its menu of variables).

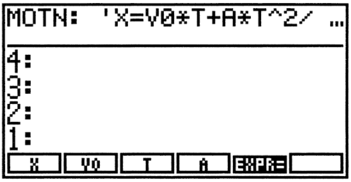
Get the Equation Catalog.

(⬅)(SOLVE) CAT



17 Move the pointer to *MOTN*. Then make it the current equation and display its menu of variables.

(▼) (if necessary)
SOLVR



Note that the selected entry doesn't become the current equation until you press SOLVR (or PLOTR).

Summary of SOLVE Menu Operations

The SOLVE Menu

Key	Programmable Command	Description
⬅ SOLVE:		
SOLVR		Selects the menu of variables for the current equation.
ROOT	ROOT	Solves an equation (in level 3) for an unknown (in level 2), using the guesses in level 1. ROOT is principally useful in programs.
NEW		Takes the equation from level 1, prompts for a variable name, stores the equation in that variable, and makes the equation in that variable the current equation.
EDEQ		Places the current equation in the command line for editing. Press ENTER to store the changes in the variable and make the edited version the current equation—or press ATTN to discard the changes.
STEQ	STEQ	Stores the level 1 equation as the current equation.
➡ STEQ	RCEQ	Recalls the current equation to level 1.
CAT		Selects the Equation Catalog.
⬅ REVIEW		Redisplays the "current equation" status message.

17

17

Solving the Current Equation

When you solve the current equation, you store values for known variables and solve for the value of the unknown variable. You normally do this using the SOLVR menu, which is a menu of variables for the current equation.

To begin solving the current equation:

- Press SOLVR in the SOLVE menu or Equation Catalog.

  or
- Press (→)(SOLVE) at any time.

In the Equation Catalog, SOLVR also sets the current equation to the selected equation in the list.

When you begin solving the equation, you get the SOLVR menu, the menu of variables for the current equation—the equation stored in *EQ*. The SOLVR menu contains:

- A "white" menu key for each variable in the current equation.
- The EXPR= menu key, discussed later in this section.

The variable menu labels are white with black letters. This emphasizes that the menu operations for variable keys in the HP Solve SOLVR menu differ from the operations in the VAR menu (or CST menu).

Finding a Solution

When you solve the current equation, the HP Solve application looks for only one solution. If the equation might have more than one solution, see "Finding Other Solutions" on page 17-17.

To solve the current equation:

1. For each variable with a known value, enter the value and press the variable's key in the SOLVR menu.
2. Optional: For the unknown variable, enter a "guess" for its value and press the variable's key in the SOLVR menu.
3. Press (←) and the unknown variable's key in the SOLVR menu to solve for its value.

If a variable doesn't exist, it's created when you store a known value or solve for its value. You don't need to enter a value for a variable if it already contains the desired value.

If you want to enter a "guess" for a variable, see "Finding Other Solutions" on page 17-17.

When you solve the equation, you get a message that describes the outcome of the process. See "Checking the Solution" on page 17-16 and "Interpreting Results" on page 17-18.

To recall a variable's value:

- Press (↱) and the variable's key in the SOLVR menu. 17

To review variable values:

1. Press (↰)(REVIEW).
2. Press (ATTN) to return to the stack display.

Example: Assuming the current equation is the equation of motion of an accelerating body (*MOTN*), calculate the distance (x) a body travels in 4 seconds (t) if its initial velocity (v_0) is 2 m/s and it is accelerating (a) at 3 m/s^2. (This equation actually has two solutions for x—to find the second solution, see "Finding Other Solutions" on page 17-17.)

(This example assumes that variables *X*, *V0*, *T*, and *A* do not exist in the current directory.)

If necessary, get the SOLVR menu. Then store 4 in *T*. (`T: 4` at the top of the display tells you 4 has been stored in *T*.)

(↱)(SOLVE) (if necessary)
4 [T]

```
T: 4
```

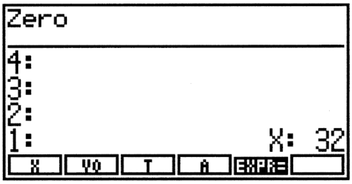
Now, store 2 in *V0* and 3 in *A*.

2 [V0]
3 [A]

```
A: 3
```

Solve for x. Note that the numeric result is tagged with the variable name. The message ZERO in the status area indicates that a root (solution) has been found.

↰ [X]

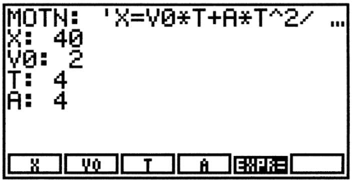


If the object actually traveled 40 meters with the same initial velocity and time, what was its acceleration? Store the new values and solve for a.

40 [X]
↰ [A]

Note that the solution for X from the previous calculation is in level 2—this is *not* the current value of X. Review the current values.

↰ (REVIEW)



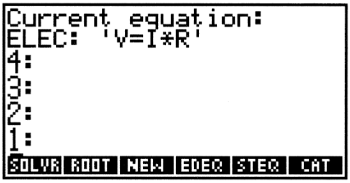
Press (ATTN) to return to the stack.

Example: The equation for a simple resistive circuit is $V = IR$, where V is the circuit voltage, I is the circuit current, and R is the circuit resistance. Use the HP Solve application to find the value of I when V is 10 volts and R is 20 ohms.

(This example assumes that variables V, I, and R do not exist in the current directory.)

Select the HP Solve application, then key in the equation. Use NEW to name the equation *ELEC* and make it the current equation.

(→) (CLR)
(←) (SOLVE)
(') V (←) (=) I (×) R
NEW ELEC (ENTER)



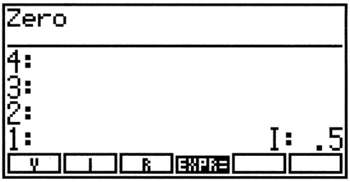
Display the menu of variables for the current equation.

SOLVR

17

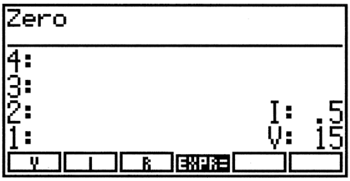
Supply the known values and solve for the unknown. The resulting message Zero in the status area indicates that the equation balances exactly at the root.

10 V
20 R
(←) I



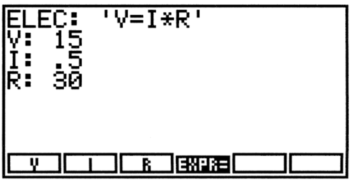
If *R* is 30 ohms for the same value of *I*, what is *V*? Store the new value of *R*, then solve for *V*.

30 R
(←) V



Now review the values of all the variables.

(←) (REVIEW)



Press (ATTN) to return to the stack display.

Checking the Solution

After you find a solution to the equation, you can check the solution by finding how closely the equation balances—how close the solution is to a true solution, called a *root*.

To check how well the solution satisfies the equation:

- Press EXPR=.

EXPR= returns one or two values, depending on the form of the current equation:

- For an equation, EXPR= returns two values—the numeric values of the left and right sides of the equation. The values are tagged with LEFT and RIGHT.
- For an expression or program, EXPR= returns one value—the numeric value of the expression or program. It's tagged with EXPR.

For an expression, the closer the result returned by EXPR= is to zero, the more likely it is that the HP Solve application has found a root. For an equation, the closer the two results returned by EXPR= are to each other, the more likely it is that the HP Solve application has found a root. For more information, see "Interpreting Results" on page 17-18.

Example: Assuming that *ELEC* is still the current equation, use EXPR= to evaluate the two sides of the equation.

Select the SOLVR menu directly and evaluate the equation.

(↱)(SOLVE)
EXPR=

The left and right sides of the equation are both exactly 15, indicating the HP Solve application found a root.

Finding Other Solutions

When you solve for a variable using the HP Solve application, it finds only one solution—even if other solutions exist. If you've already found a solution for your equation and you want to find another solution, you can use guesses to guide the root-finder to a different solution. See "Using Guesses" below.

For a more visual approach to finding solutions, you can use the Plot application. See "Choosing the HP Solve or Plot Application" on page 17-22.

Using Guesses

17

You can supply one or more guesses for the unknown variable before solving for it. Good guesses help in two ways:

- If there's more than one solution, guesses control which solution is found.
- Good guesses reduce the time required to find a solution.

To store guesses for a variable:

- To store one guess, enter the value and press the unshifted menu key.
- To store two or three guesses to bracket a desired solution, enter a list (with { } delimiters) containing the guesses, then press the unshifted menu key.

To find out how guesses affect the solution, see "How the Root-Finder Uses Initial Guesses" on page 17-32.

Example: Store guesses 0 and 10 for variable V. Press [↰] [{ }] 0 [SPC] 10 [ENTER] [V] to create the list and store it in V.

Summary of SOLVR Menu Operations

The SOLVR menu normally contains a label for each global variable in the current equation plus the EXPR= label. You can customize this menu—see "Customizing the SOLVR Menu" on page 17-25.

To use the SOLVR menu:

- To store a value in a variable, enter the new value on the stack, then press its menu key.
- To solve for a variable's value, press (⬅) and its menu key.
- To recall a variable's value, press (➡) and its menu key.
- To type a variable name when the command line is in Algebraic- or Program-entry mode, press the menu key.
- To calculate the "value" of the equation, press EXPR=.
- To review variable values, press (⬅)(REVIEW). (Press (ATTN) to return to the stack display.)

17

The SOLVR menu remains unchanged until a new current equation is specified.

The catalog of values shows the full names and values of variables on the current page of the SOLVR menu. The next keystroke you make cancels the review, redisplays the stack, and then executes the keystroke itself.

Interpreting Results

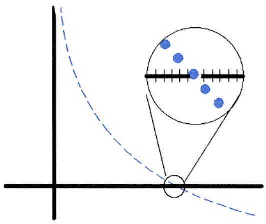
The HP Solve application returns a message describing the result of the root-finding process. You can use this message and other information to judge whether the result is a root of your equation.

The message is based on the *value of the equation*—the difference between the left and right sides of an equation, or the value returned by an expression or program.

When a Solution is Found

If a root is found, the HP Solve application returns a message describing the root:

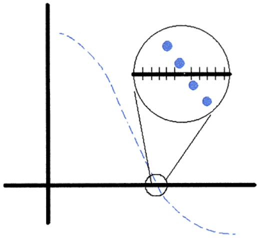
`Zero` The HP Solve application found a point where the value of the equation is 0 within the calculator's 12-digit precision.

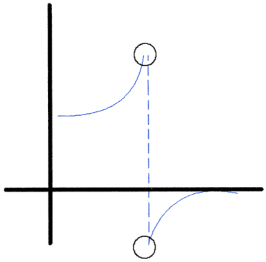


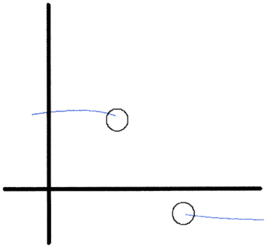
`Sign Reversal` The HP Solve application found two points where the value of the equation has opposite signs, but it cannot find a point in between where the value is 0. This may be because:

- The two points are neighbors (they differ by 1 in the 12th digit).
- The equation is not real-valued between the two points. The HP Solve application returns the point where the value is closer to 0. If the value of the equation is a continuous real function, this point is the HP Solve application's best approximation of an actual root.

17

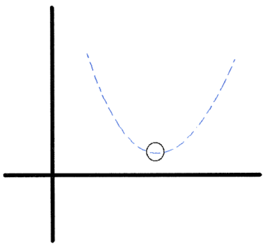






`Extremum` One of the following occurred:

- The HP Solve application found a point where the value of the equation approximates a local minimum (for positive values) or maximum (for negative values). The point may or may not represent a root.
- The HP Solve application stopped searching at ±9.99999999999E499, the largest or smallest numbers in the calculator's range of numbers.



To obtain more information, you can:

- Evaluate the equation using EXPR=. For an expression or program, the closer the result is to 0, or for an equation, the closer the two results are to each other, the more likely it is that the HP Solve application found a root. You must use judgement in considering the results.
- Plot the expression or equation in the region of the answer. The Plot application will show any local minimum, maximum, or discontinuity.
- Check the system flags that detect mathematical errors (see appendix E). For example, flag −25 indicates whether overflow occurred.

17

When No Solution is Found

If the HP Solve application can't return a result, it displays a message indicating the reason:

Bad Guess(es)	One or more of the initial guesses lie outside the domain of the equation—or units for the unknown variable aren't consistent with the units for the other variables (see "Using the HP Solve Application with Unit Objects" on page 17-23). Therefore, when the equation was evaluated, it didn't return a real number or it generated an error.
Constant?	The value of the equation is the same value at every point sampled.

17

Choosing the HP Solve or Plot Application

The HP Solve application and the Plot application both let you find solutions for an equation.

The Plot application lets you find solutions by working directly with a picture of the equation. This powerful capability is of great value if you don't know what the equation looks like over a range of values. Specifically, equations can have multiple solutions—and they can have local minima and maxima.

With the HP Solve application, you use numeric guesses to direct the root-finder to the desired region of the equation. The Plot application lets you graph the equation and then move the special graphics cursor directly to the region of the equation that contains the desired solution. (See "Analyzing Functions" on page 18-25 for more information.)

Advantages of HP Solve Application	Advantages of Plot Application
■ You can easily store values for the known variables and solve for the unknown, and you can easily change which variable is the unknown. ■ You can review the values of the variables in the equation. ■ You can use unit objects requiring automatic unit conversion.	■ You can see if an equation has multiple solutions or local extrema. ■ You can direct the application to a specific solution simply by moving a cursor, rather than by entering numeric guesses.

Using the HP Solve Application with Unit Objects

The current equation and any of its variables may contain unit objects. The SOLVR menu processes unit objects automatically.

To use units with the SOLVR menu:

- To store a value *with units* in a variable, enter the unit object, then press the variable menu key.
- To change a variable's value *and keep its old units*, enter the number only, then press the variable menu key.
- To solve for a variable:
  1. Enter one or more guesses *with the desired units*, then press the variable menu key.
  2. Press ⬅ and the variable menu key to find the solution.

Keep these guidelines in mind:

- Before solving, all variables must contain a consistent set of units—*including the unknown variable*. For example, if the equation is 'Y=X/T' and you've stored 2_m in X and 3_s in T, you must enter a guess for Y with the dimensions *length/time*. The solution is automatically converted to the units specified in the guess. If you enter a guess of 1_ft/yr, the solution will have units of ft/yr.
- If you're entering a list of two or three guesses, one of the guesses must have the appropriate units. (If more than one guess has units, the units of the last guess and only the number parts of the other guesses are used.)
- If the equation you're solving uses or calculates temperature *difference* (as opposed to actual temperature *level*), use K or °R (not °C or °F). For temperature *conversion*, use the UNITS catalog menu.

Note

Because the SOLVR menu allows you to change the number portion of a unit object without affecting the unit, you must purge variables containing unit objects before using them in equations requiring numbers only.

17

Example: Use the equation $C = Q/V$ to calculate the capacitance C when $Q = 8.9 \times 10^{-6}$ coulombs and $V = 57$ volts.

(This example assumes that variables C, Q, and V do not exist in the current directory.)

Enter and name the equation, then select the SOLVR menu.

⮫ CLR
⮪ SOLVE
' C ⮪ = Q ÷ V NEW
CAP ENTER
SOLVR

```
CAP: 'C=Q/V'
4:
3:
2:
1:
 C  Q  V  EXPR=
```

17

Enter the known values, then store them.

⮪ UNITS NXT ELEC
57 V
8.9 EEX 6 +/– C
⮫ SOLVE Q
V

```
V: 57_V
```

Store a guess in farads, then solve for the unknown.

⮫ LAST MENU 1 F
⮫ LAST MENU C
⮪ C

```
Zero
3:
2:
1: C: 1.56140350877E-7
    _F
 C  Q  V  EXPR=
```

Now change the problem: Solve for V in millivolts for $C = 22$ picofarads and $Q = 1.7 \times 10^{-10}$ coulombs. Store the new value of Q—you don't need to append the unit. Store C with its new unit.

1.7 EEX 10 +/– Q
22 ⮫ _ pF ENTER C

```
C: 22_pF
```

Store a guess for V in millivolts and solve for the unknown.

1 ⮫ _ mV ENTER V
⮪ V

```
Zero
4:
3:
2: C: 1.56140350877E-…
1: V: 7727.27272726_mV
 C  Q  V  EXPR=
```

Customizing the SOLVR Menu

You can customize the SOLVR menu—so you can specify and solve equations *and* perform other calculator operations without leaving the SOLVR menu:

- Specify the equation to solve.
- Define which equation variables appear in the menu and their order.
- Include other objects in the menu that you can execute.

To create a customized SOLVR menu:

17

1. Enter a *solver-list* (with { } delimiters) in level 1. (The content of the list is described below.)
2. Press (↰)(SOLVE) NEW , enter a name, and press (ENTER).

The equation in the list becomes the current equation. The syntax of the solver-list is

{ *equation* { *key-definitions* } }

where

equation	Specifies the equation. It can be an equation or expression (with ' delimiters), a program object (with « » delimiters), or the name of an equation, expression, or program.
key-definitions	Specifies the menu keys—each entry defines one key. Each entry can be either a variable name (with SOLVR behavior) or other type of object (with CST behavior). (To include a program that you can execute, either enter its *name* in the key-definition as a sublist of the form { "*label*" « *name* » }, or enter the program object itself in the key-definition.)

The CST behavior for various object types in the key-definition list is described under "Creating a Custom Menu" on page 15-1.

To include a custom SOLVR menu in the Equation Catalog:

- Enter the solver-list, then use `NEW` to name it.

  or
- Store the solver-list in a variable whose name ends in `EQ`.

The Equation Catalog includes all algebraic objects and all variables whose name ends in `EQ`. `NEW` automatically adds `.EQ` to the name when the level 1 object is a list or program.

Example: The equation $I = 2\pi^2 f^2 \rho v a^2$calculates the intensity of a sound wave. Suppose you always calculate the value of ρ and store it in the corresponding variable *prior* to using this equation, and so would like to suppress ρ from the SOLVR menu.

17

The solver-list

```
{ 'I=2*π^2*f^2*ρ*v*a^2' { I f v a } }
```

when stored in *EQ*, creates this SOLVR menu for the equation

```
[ I ] [ F ] [ V ] [ A ] [EXPR=] [ ]
```

and suppresses ρ from the menu. To save this solver-list in the equation catalog, store it in a variable ending in EQ, for example, *I.EQ*.

Example: Suppose you want the IP command available in the SOLVR menu so that you can store integer values in the variables in the SOLVR menu. The following solver-list list amends the solver-list in the previous example to include two additional keys: a blank key and a key that executes IP (integer part).

```
{ 'I=2*π^2*f^2*ρ*v*a^2' { I f v a { } IP } }
```

The list, when stored in *EQ*, creates this menu of variables and functions:

```
[ I ] [ F ] [ V ] [ A ] [ ] [ IP ]
```

Solving Two or More Equations

You may often work with two or more related equations—for example, equations with common variables. Putting several equations in a list lets you share known and solved values among those equations—and easily change the equation you're solving.

To create a list of two or more equations:

1. Press ⬅ SOLVE CAT or ➡ ALGEBRA to get the Equation Catalog.
2. Press ▼ and ▲ to move the pointer to an equation you want to include.
3. Press EQ+ to add the equation to the list. (If necessary, press ⬅ EQ+ to remove the last entry from the list.)
4. Repeat steps 2 and 3 for each equation you want to include.
5. Press SOLVR to begin solving the first equation in the list.

17

Each time you press EQ+ , a list containing the selected equations is displayed and updated in the status area.

When you press SOLVR, the list is stored in *EQ* and the SOLVR menu for the first equation in the list is displayed, with the additional key NXEQ .

If the current equation is a list of equations, the equations are "linked"—you can easily switch the SOLVR menu from one equation to another.

To begin solving the next equation:

- Press NXEQ .

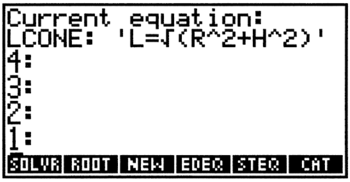
The equation names rotate in the list, moving the second name to the beginning of the list—and the variables for that equation appear in the SOLVR menu.

Example: Create the two equations: 'L=√(R^2+H^2)' and 'V=π*R^2*H/3'. Use NEW to name them *LCONE* and *VCONE* respectively, and to store them in the Equation Catalog. Then put the two equations in a list and find the radius of a right circular cone whose height is 10 meters and whose slant height is 25 meters. Then find its volume.

(This example assumes that variables L, R, H, and V do not exist in the current directory. If you used V in the circuit example on page 17-14 and you're now in the same directory, you'll need to purge V.)

Key in and store the first equation.

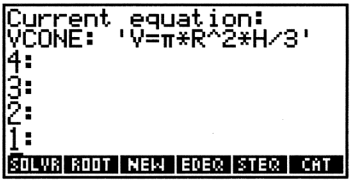
[↱] [CLR]
[↰] [EQUATION]
L [↰] [=] [√x] [↰] [()]
R [y^x] 2 [▶] [+] H [y^x] 2 [ENTER]
[↰] [SOLVE] NEW LCONE [ENTER]



17

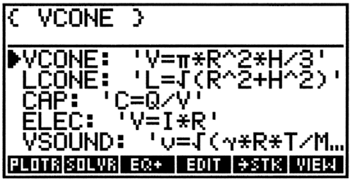
Key in and store the second equation.

[↰] [EQUATION]
V [↰] [=] [↰] [π] [×]
R [y^x] 2 [▶] [×] H [÷] 3
[ENTER]
NEW VCONE [ENTER]



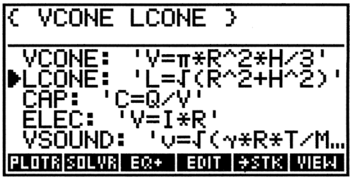
Put the two equations in a list and start solving *VCONE*. Press [▲] or [▼] as required to position the cursor at equation *VCONE*.

CAT
([▲] or [▼] if necessary)
EQ+



Move the cursor to *LCONE* and execute EQ+.

[▼] EQ+

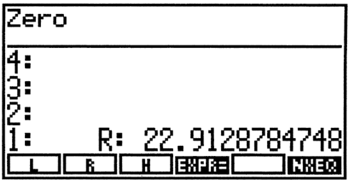


Start solving the linked equations.

SOLVR

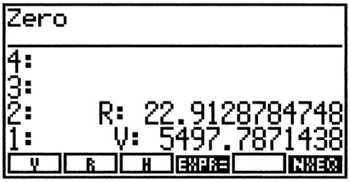
Display the menu of variables for *LCONE*. Then supply values for the known variables and solve for the unknown radius.

NXEQ 10 H
25 L
⬅ R



Switch to the menu of variables for *VCONE*. You can solve directly for the volume since the radius and height are already stored in their variables.

NXEQ ⬅ V



Simply storing a list of equations in *EQ* doesn't name the list. If you don't name the list, the list is lost if you later change *EQ*. NEW adds .EQ to the name so the list is included in the Equation Catalog.

To name a list of equations currently in EQ:

1. Press ⬅ SOLVE ➡ STEQ (the RCEQ command) to recall the list to level 1.
2. Press NEW to name the list.

To create and name a list of equations:

1. Press ⬅ SOLVE CAT or ➡ ALGEBRA to get the Equation Catalog.
2. Use EQ+ to add the desired equations to the list.
3. Press →STK to copy the list onto the stack.
4. Press ATTN to leave the Equation Catalog.
5. Press ⬅ SOLVE NEW to name the list.

Finding the Solution of a Program

The HP Solve application accepts a program as the current equation. Using a program as the current equation is useful when the relationship between variables can't be written symbolically. The solution is a value of the unknown variable for which the program returns a value of zero.

To design a program for the HP Solve application:

- The program must take nothing from the stack.
- The program must return only one result.

Example: The UTPC (*upper tail chi-square distribution*) command in the MTH PROB menu calculates the probability that a chi-square random variable with n degrees of freedom is greater than χ. The relationship is

$$UTP = \mathrm{UTPC}(n, \chi^2)$$

where UTP is the unknown variable.

However, the UTPC command can't be included in a symbolic equation. But the relationship can be rewritten as an expression that should equal 0:

$$UTP - \mathrm{UTPC}(n, \chi^2)$$

This program computes the value of the expression:

```
« UTP N CHI2 UTPC - »
```

Use this program to calculate the upper tail probability (UTP) for $CHI2$ = 6.2 and N = 5. Then calculate χ^2 to a significance of 0.1 (UTP = 0.1) for 5 degrees of freedom.

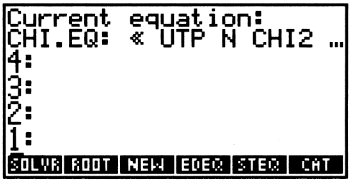
Enter the program.

(→) (CLR)
(←) («») UTP N CHI2
(MTH) PROB (NXT) UTPC (−)
(ENTER)

```
1: « UTP N CHI2 UTPC -
   »
UTPC | UTPF | UTPN | UTPT
```

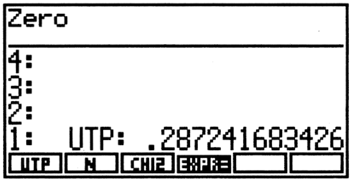
Name the program and make it the current equation. Note that NEW automatically prompts you with .EQ since the object is a program.

⬅ SOLVE NEW
CHI ENTER



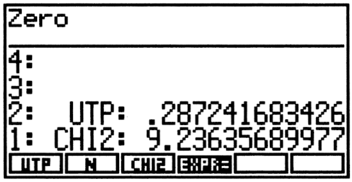
Display the SOLVR menu and store the known values. Then calculate the upper tail probability.

SOLVR
5 N
6.2 CHI2
⬅ UTP



Now store the significance in variable *UTP* and solve for *CHI2*.

.1 UTP
⬅ CHI2



How the HP Solve Application Works

Pressing a left-shifted menu key in the SOLVR menu activates the numeric *root-finder*, which seeks a solution iteratively. Starting with the guesses you've stored in the variable, or the guesses that the calculator itself provides, it generates pairs of intermediate guesses until a solution is found. The HP 48 displays Solving for ... while the root-finder is executing.

In searching for a solution, the root-finder seeks a value of the unknown for which the value of the expression equals 0. (Equations are treated internally as expressions of the form '*left-side−right-side*'.) First, the root-finder searches for two points where the expression's value has opposite signs. When it finds a sign reversal, the root-finder tries to narrow the search region until it finds a point where the expression's value is 0.

How the Root-Finder Uses Initial Guesses

You can enter one, two, or three values as guesses. Two or three values are entered as a list.

- **One value**. The number is converted to two values by copying the number and adding a small perturbation to one copy.
- **Two values**. The numbers identify a region where the search will begin. If the two guesses yield expression values with opposite signs, the root-finder usually finds a root between the two numbers rapidly. If the two guesses yield expression values with the same sign, the search generally takes longer.
- **Three values**. The first number should be your best guess for the root. The second and third numbers are used as two values, above.

Halting the Root-Finder

To halt the root-finder:

- Press (ATTN).

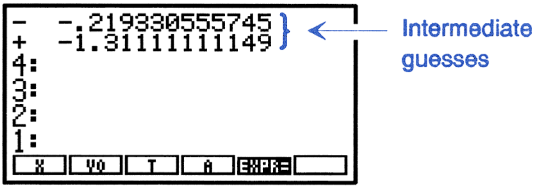
The HP Solve application returns a list containing three values: the best value found so far plus two values that identify the region that was being searched.

To restart the root-finder:

1. Put a list of three values in level 1:
   - To restart from where it left off, use the list left by the root-finder.
   - To restart in a different region, enter a different list.
2. Press the menu key for the unknown variable to store the list.
3. Press (←) and the menu key for the unknown variable.

Displaying Intermediate Guesses

While the HP 48 is displaying the `Solving for` message, pressing any key except (ATTN) displays pairs of intermediate guesses and the sign of the values of the expression for each guess. If the expression is undefined at the guess, `?` is shown.



Watching the intermediate guesses can give you information about the root-finder's progress—whether the root-finder has found a sign reversal (the guesses have opposite signs), or if it is converging on a local minimum or maximum (the guesses have the same signs), or if it is not converging at all. In the latter case, you may want to halt the root finder and restart with a new guess.

How the Menu of Variables Is Created

The menu of variables contains a label for each variable in the current equation. If the variable does not already exist, it is created and added to the current directory when you store a value in it.

If a variable in the current equation contains an algebraic object (or a name or program), the variable itself is not included in the menu of variables. Instead, the variables in the algebraic object are used.

For example, if the current equation is `'A=B+C'`, and *B* contains the expression `'D+TAN(E)'`, the menu of variables is:

A	D	E	C	EXPR=	

17

Note that for equations that contain a *where* clause (see "The | (Where) Function" on page 22-25) or an integral, summation, or derivative, the *placeholder* variable appears in the SOLVR menu. For example, the SOLVR menu for the equation

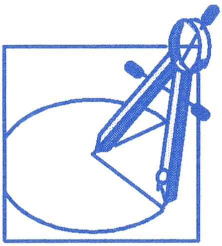
$$A + B - \int_0^1 2X dX = 0$$

will contain a key labeled X. However, you *cannot* solve for this placeholder variable.

17

18

Basic Plotting and Function Analysis



The Plot application lets you draw graphs of one or more functions in various formats, calculate roots and other parameters, plot statistical data in various formats, and embellish plots with additional elements.

To plot a mathematical function by hand you would use the following general procedure:

1. Write down the function you want to plot.
2. Select the independent variable, for example x, in the function. Then determine the range of x-values to plot and the number of (evenly spaced) sample points. From this information, draw an appropriately scaled x-axis. Then draw an appropriately scaled y-axis based on your estimates of the function's value over the plotted interval.
3. For every value of x, calculate the value of the function $f(x)$, and plot the corresponding point $(x, f(x))$.
4. Draw a smooth curve through the points.

When you use the Plot application, you follow a similar procedure, as you'll see in this chapter.

This chapter covers basic plotting and analysis of mathematical functions—how to specify the current equation, how to specify plot parameters, and how to analyze function plots. (All examples in this chapter use the FUNCTION plot type.)

Chapter 19, "More about Plotting and Graphics Objects," builds on these concepts, giving information about other plot operations and about graphics objects.

The Structure of the Plot Application

You can use the Plot application to plot functions represented by equations (or by expressions or programs). The Plot application contains special data elements that parallel the elements of the procedure described above:

18

- Reserved variable *EQ* contains the equation you want to plot. The equation in *EQ* is called the *current equation*. Note that *EQ* is also used by the HP Solve application to build the SOLVR menu.
- Reserved variable *PPAR* contains specifications for the independent variable, the display and plotting ranges, the number of sample points in the plotting range, and the axes.
- *PICT*, a part of HP 48 memory, is analogous to the piece of paper on which the plot is drawn.

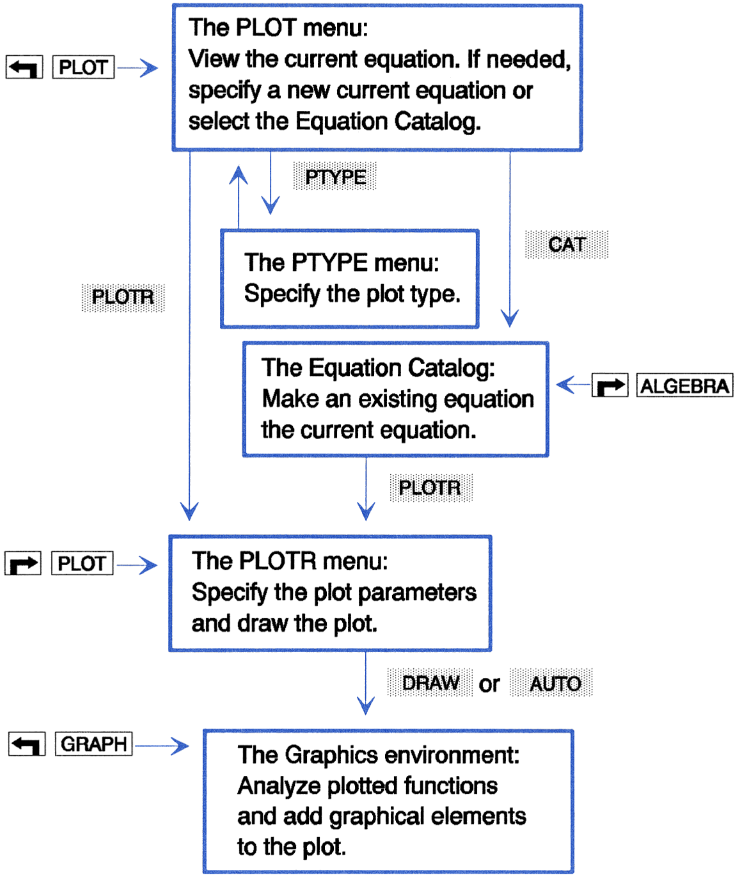
These data elements are tied to two menus and a special environment:

- The PLOT menu is used for the selection or modification of the current equation. The PLOT menu is also used to specify the *plot type*, which determines how the HP 48 interprets the equation. For example, the equation may represent a conic section—in this case, the appropriate plot type is CONIC.
- The PLOTR menu is used to specify the contents of *PPAR* and to draw the plot.
- The *Graphics environment* is used to view the graph, analyze the mathematical behavior of the plot, and add graphical elements to it.

In general, you use these steps to plot an equation with the Plot application:

1. Use the PLOT menu to store the equation in *EQ* and, if necessary, to specify the plot type.
2. Use the PLOTR menu to set the appropriate plot parameters.

3. Draw the graph.
4. Use the operations in the Graphics environment to obtain data from the graph or add graphical elements to it.



18

In the Graphics environment, the display shows the contents of *PICT*, and the keyboard is redefined to execute graphics operations. When the HP 48 finishes a plot, it automatically puts you in the Graphics environment. If you switch back to the stack display, *PICT* persists—you can reenter the Graphics environment at any time to view *PICT*.

In the Graphics environment you do not have access to the stack. However, function analysis operations in the Graphics environment return their results to the stack. In addition, all or parts of *PICT* can be copied to the stack as an object called a *graphics object*. Commands in the PRG DSPL menu let you work with graphics objects on the stack and let you move a graphics object back into *PICT*.

Using Equations, Expressions, and Programs

The HP 48 can plot an equation, expression, or program:

- **Equation.** An equation is an algebraic object containing = (for example, `'A+B=C'`).
- **Expression.** An expression is an algebraic object not containing = (for example, `'A+B+C'`).
- **Program.** A program to be plotted must return one real number.

Throughout this chapter, unless otherwise stated, the term "equation" refers to all objects used to create plots: equations, expressions, programs, and lists of equations, expressions, and programs.

You can also plot statistical data—see "Plotting Statistical Data" on page 21-13.

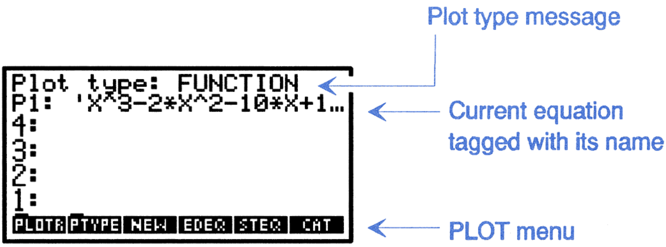
Specifying the Current Equation and Plot Type

The *current equation* is the equation you last solved or plotted. It's stored in reserved variable *EQ*. You change the current equation each time you solve or plot a different equation.

To check the current equation and plot type:

- Press [↰] [PLOT].

A two-line status message gives the current equation and the plot type—or, if there's no current equation, it gives instructions for entering a new equation. In addition, the PLOT menu is displayed.



Changing the Current Equation and Plot Type

How to specify the current equation and use the Equation Catalog is covered in detail in "Specifying the Current Equation" on page 17-3. Only certain instructions are repeated below.

To enter and name a new current equation:

1. Enter the equation in level 1. You can type it in the command line or use the EquationWriter application.
2. Press [←][PLOT] NEW .
3. Without pressing [α], key in a name for the equation and press [ENTER].

To change the plot type:

1. Press [←][PLOT]
2. Press PTYPE.
3. Press a menu key to select one of the eight plot types.

You can also change the plot type during the next stage, when you're setting the plot parameters using the PLOTR menu.

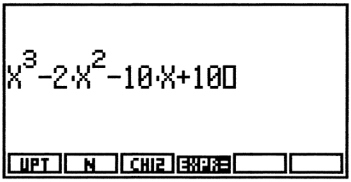
To select and plot an equation from the Equation Catalog:

1. Press [←][PLOT] CAT .
2. Press [▲] and [▼] to move the pointer to the desired entry in the list.
3. To make the equation the current equation (and to start setting up and drawing the plot), press PLOTR.

Example: Set the current equation to the expression $x^3 - 2x^2 - 10x + 10$ and the plot type to FUNCTION.

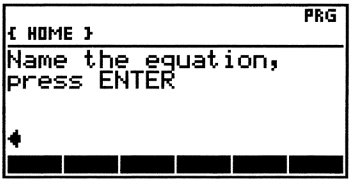
Key in the expression using the EquationWriter application.

[↱] [CLR]
[↰] [EQUATION] X [y^x] 3 [▶] [−]
2X [y^x] 2 [▶] [−]
10X [+] 10



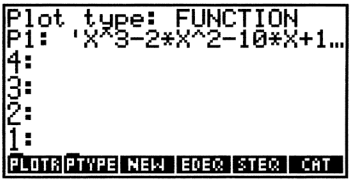
Store the equation as the current equation.

[ENTER]
[↰] [PLOT] NEW



The HP 48 prompts you to enter a variable name and activates the alpha keyboard. Enter the name *P1*. Then set the plot type to FUNCTION if necessary.

P1 [ENTER]
(PTYPE FUNC if necessary)



Summary of PLOT Menu Operations

The PLOT Menu

Key	Programmable Command	Description
◀ PLOT:		
PLOTR		Selects the PLOTR menu for specifying the plot parameters in *PPAR* and for drawing the plot.
PTYPE		Displays the PTYPE menu for specifying the plot type.
NEW		Takes an equation from level 1, prompts for a variable name, stores the equation in that variable, and makes the equation in that variable the current equation.
EDEQ		Places the current equation in the command line for editing. Press ENTER to store the changes in the variable and make the edited version the current equation—or press ATTN to discard the changes.
STEQ	STEQ	Stores the level 1 equation as the current equation.
▶ STEQ	RCEQ	Recalls the current equation to level 1.
CAT		Selects the Equation Catalog.
◀ REVIEW		Redisplays the "current equation" status message.

18

Setting Plot Parameters and Drawing the Plot

When you plot the current equation, you first set up the plot by specifying the independent variable and scaling, then draw the plot. You normally do this using the PLOTR menu.

To begin setting up the plot:

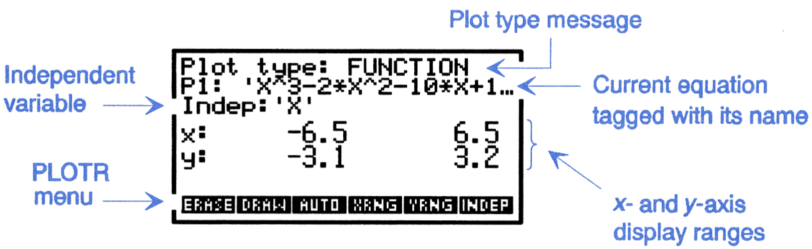
- Press PLOTR in the PLOT menu or Equation Catalog.

  or
- Press (↱)(PLOT) at any time.

18 In the Equation Catalog, PLOTR also sets the current equation to the selected equation in the list.

The PLOTR menu display includes a status message describing:

- The *plot type* (discussed under "Choosing Plot Types" on page 19-12).
- The current plot data—either the *current equation* or the *current statistical data*—if there is any.
- The *independent* variable and, if specified, the *plotting* range (discussed under "Using Plotting Range instead of Display Range" on page 19-1).
- The *display* ranges in the horizontal and vertical directions. In this message, x always indicates the horizontal direction, and y always indicates the vertical direction.



Specifying the Independent Variable

For function, polar, and parametric plots, the only variable name you have to specify is for the *independent* variable. If the current independent variable isn't the one you need, you can change it. The default independent variable is X.

To change the independent variable:

1. Enter the variable name (with ' delimiters).
2. Press INDEP in the PLOTR menu.

The role of the independent variable in building function plots is discussed in more detail under "How DRAW Plots Points" on page 18-17.

18

Example: If you want to plot 'S=4*T^2+6', you must specify the new independent variable T. In the PLOTR menu, press ['] T INDEP.

Setting the Display Ranges or Scaling

You can define the ranges of values represented by the plotting area in either of two ways:

- **Display ranges.** This lets you directly set the extreme limits of the plotting area.
- **Scales and center.** This lets you directly set the intervals represented by the tick marks along the axes and the coordinates located at the center of the plot.

Regardless of which method you use, your specifications are shown in the PLOTR menu display as display ranges, and they're stored in *PPAR* as display ranges.

To set one or both display ranges:

- To change the horizontal range, enter the two limits for the x-axis (press [SPC] or [ENTER] to separate the numbers), then press XRNG in the PLOTR menu.
- To change the vertical range, enter the two limits for the y-axis (press [SPC] or [ENTER] to separate the numbers), then press YRNG in the PLOTR menu.

The horizontal and vertical display ranges are the ranges of values represented by the plotting area *PICT*. If the current display ranges aren't the ones you need, you can change them. The default display range along the x-axis is −6.5 to 6.5 units, and along the y-axis is −3.1 to 3.2 units.

You may not have to specify the vertical display range—the y-axis display range is computed for you if you plot the graph with automatic scaling (using the AUTO command described later).

Example: Specify a display range along the x-axis from −10 to 40 units. In the PLOTR menu, press 10 [+/-] [SPC] 40 XRNG.

18

To set the scales or center:

- To set the scales, enter the x-axis tick interval, press [SPC] or [ENTER], enter the y-axis tick interval, and press SCALE in the PLOTR menu.
- To set the center, enter the (x,y) coordinates as a complex number (press [←][( )] x [SPC] y), then press CENT in the PLOTR menu.

The scales of the x- and y-axes represent the number of units per tick mark along the axes. You can use SCALE if you want the axes tick marks to represent meaningful values (such as integer values) or if you want equal scaling for the two axes. The default x and y scales are 1 and 1.

The center point of the display is specified by a complex number representing its coordinates. You can use CENTR if you want to see a certain region of the graph. The default center is (0,0).

Example: Make each x-axis tick mark represent 2 units and each y-axis tick mark represent 5 units. Make coordinates (40,50) be located in the center of the display. Press 2 [SPC] 5 SCALE and [←][( )] 40 [SPC] 50 CENT.

Resetting Plotting Parameters

You can reset all plotting parameters except the plot type to their default values. (This also erases the plotting area *PICT* and restores its default size.)

To reset plotting parameters:

- Press RESET in the PLOTR menu.

Drawing the Graph

After you've set the plotting parameters, you're ready to draw the graph. You can draw it in either of two ways:

- **Autoscaling the y-Axis.** This lets you draw a graph when you're not sure of the appropriate y-axis display range. The vertical display range is determined by sampling the equation across the x-axis display range.
- **Specifying the y-Axis.** This lets you preserve the vertical range or scaling you've specified.

To draw the graph with autoscaling:

- Press AUTO in the PLOTR menu.

For function plots, AUTO evaluates the equation at 40 values spaced equally across the range of the independent variable, computes the vertical display range, and draws the graph (using DRAW). AUTO also erases the previous plot in *PICT*.

To draw the graph with the specified range:

- Optional: To erase the existing plot, press ERASE in the PLOTR menu.
- Press DRAW in the PLOTR menu.

DRAW is faster than AUTO because it doesn't sample the equation.

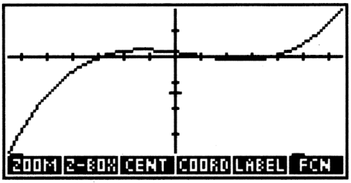
Example: Plotting with AUTO. Plot the previous equation *P1* using autoscaling and the default plot parameters. (The independent variable is *X*, which is the default name.)

Get the PLOT menu and make sure you're using equation *P1*.

(←) (PLOT) PLOTR

Reset the plot parameters and draw the graph using the default plot parameters.

(NXT) RESET (*not* RES)
(⬅)(PREV) AUTO



Press (ATTN) to return to the stack display.

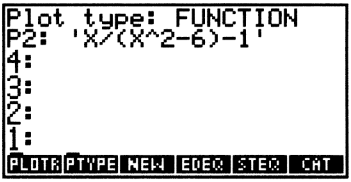
18

Example: Plotting with AUTO. Use autoscaling to plot the equation

$$\frac{x}{x^2-6}-1$$

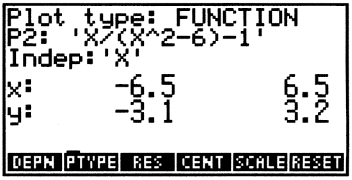
Key in the equation using the EquationWriter application. Name it *P2*.

(⬅)(EQUATION)
X (÷) X (y^x) 2 (▶)
(−) 6 (▶) (−) 1
(ENTER)
(⬅)(PLOT)
NEW P2 (ENTER)



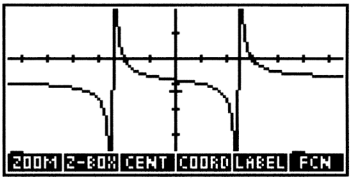
Get the PLOTR menu and reset the plotting parameters.

PLOTR (NXT) RESET



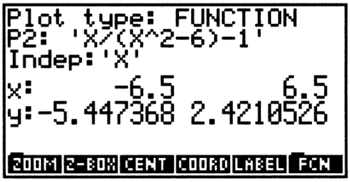
Draw the graph using autoscaling. (The vertical lines in the plot represent the connecting of points at discontinuities in the function—see the next topic, "Choosing Connected or Disconnected Plotting.")

(⬅)(PREV) AUTO



Press [⬅] [REVIEW] to check the PLOTR menu status message to see the newly computed y-axis display range. (Hold the [REVIEW] key down to keep displaying the message).

[⬅] [REVIEW] (hold)

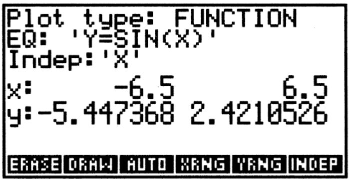


Press [ATTN] to return to the stack display.

Example: Plotting with DRAW and Ranges. Plot the equation $y = \sin(x)$. Use a display range of -5 to 5 along the x-axis and -1.1 to 1.1 along the y-axis. 18

Select Radians mode, key in the equation, and store it directly into *EQ* without naming it.

([⬅] [RAD] if necessary)
['] Y [⬅] [=] [SIN] X
[⬅] [PLOT] STEQ
PLOTR

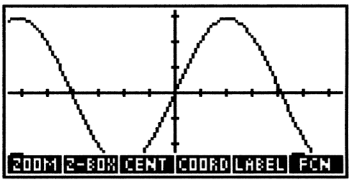


Set the display ranges.

5 [+/−] [SPC] 5 XRNG
1.1 [+/−] [SPC] 1.1 YRNG

Erase *PICT* and draw the graph.

ERASE DRAW

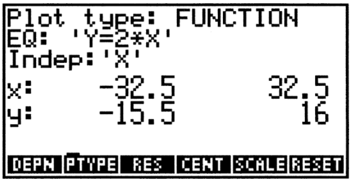


Press [ATTN] [⬅] [RAD] to return to the stack display and select Degrees mode.

Example: Plotting with DRAW and Scales. Plot the equation $y = 2x$. To make the slope (2) "look" correct, specify equal scaling for both axes, and put the origin (0,0) at the center of the display.

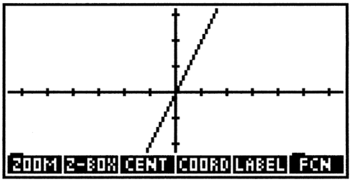
Key in the equation and store it directly into *EQ* without naming it. Select the PLOTR menu and specify the center and scale. For the scale, specify 5 units per tick mark. (Note how the display ranges are recomputed after you execute SCALE.)

['] Y [⬅] [=] 2 [×] X
[⬅] [PLOT] STEQ
PLOTR [NXT]
18 [⬅] [( )] 0 [SPC] 0 CENT
5 [SPC] 5 SCALE



Draw the graph.

[⬅] [PREV] ERASE DRAW



Press [ATTN] to return to the stack display.

Choosing Connected or Disconnected Plotting

Initially, DRAW connects successive computed points with straight line segments. The connections are made regardless of the relative positions of the plotted points. This may be graphically undesirable, such as for a function with a discontinuity. (An earlier example in this section plots a function with multiple discontinuities, and connects each point.)

To change the "connect" plotting option:

1. Press [⬅] [MODES] [NXT].
2. Press CNC■ or CNCT .

The connect option is not controlled by a plot parameter—it's controlled by a system flag, flag −31, which is initially clear. CNC■ indicates plotted points are connected (flag −31 clear). CNCT indicates points are not connected (flag −31 set).

Summary of Basic PLOTR Menu Operations

The PLOTR menu contains the basic commands for setting the plot parameters and for drawing the plot. Other PLOTR menu operations are described under "Refining Plots" on page 19-1.

The PLOTR Menu—Basic Plotting Operations

Key	Programmable Command	Description
(⬅) (PLOT) PLOTR or (➡) (ALGEBRA) PLOTR:		
ERASE	ERASE	Erases *PICT*, leaving a blank *PICT* of the same size.
DRAW	DRAW	Draws the plot using the x- and y-axis ranges. DRAW does not erase *PICT*—the plot is added to any previous contents of *PICT*. When executed from a program, DRAW does not include axes in the graph. ((⬅) DRAW executes STEQ. (➡) DRAW executes RCEQ.)
AUTO	AUTO	Draws the graph using the x-axis range, and autoscales the y-axis. Any previous plot in *PICT* is erased. When executed from a program, AUTO only autoscales the y-axis—it does not draw a graph.
XRNG	XRNG	Sets the display range of the horizontal axis using two real-number arguments—$x_{\min}$ and $x_{\max}$. ((➡) XRNG recalls the current x-axis display range.)

18

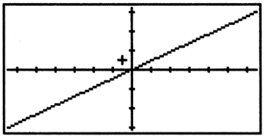
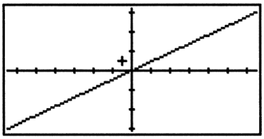
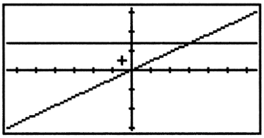
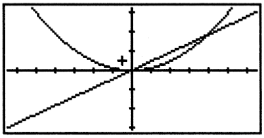
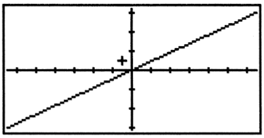
The PLOTR Menu—Basic Plotting Operations (continued)

18

Key	Programmable Command	Description
YRNG	YRNG	Sets the display range of the vertical axis using two real-number arguments—y_{min} and y_{max}. (⮫ YRNG recalls the current y-axis display range.
INDEP	INDEP	Sets the name in level 1 as the independent variable. INDEP can also specify the plotting range for the independent variable (see "Using Plotting Range instead of Display Range" on page 19-1). (⮫ INDEP recalls the current independent variable, and its plotting range if specified.)
PTYPE		Selects the PTYPE menu for changing the plot type.
CENT	CENTR	Takes a complex number (x,y) and makes it the center coordinate of the display. (⮫ CENT recalls the current center coordinate.)
SCALE	SCALE	Takes two real-number arguments. The first argument sets the x-scale in units per 10 pixels. The second argument sets the y-scale. (⮫ SCALE returns the x- and y-scales.)
RESET		Resets all plot parameters except the plot type to their default states and erases *PICT*, restoring it to its default size (131 pixels wide by 64 pixels high).
⮪ REVIEW		Redisplays the plot parameters.

How DRAW Plots Points

In this section, it's necessary to reassert the normal distinction between equations, expressions, and programs. For function plots, DRAW treats expressions and programs the same way—but it plots equations according to their structure and the setting of flag −30, as shown below.

Contents of EQ	Example	Graph
' *expression* '	`'3*X'`	
' *name=expression* '	`'Y=3*X'`	**Flag −30 clear:**  **Flag −30 set:** 
' *expression=expression* '	`'X^2=3*X'`	
« *program* »	`« 3 X * »`	

DRAW evaluates each expression it plots for a series of values of the independent variable along the x-axis range. This generates a series of points $(x, f(x))$. The number of values of the independent variable for which the expression is evaluated depends on the *resolution* (discussed in "Specifying Resolution" on page 19-3).

For function plots, the currently specified *dependent* variable is ignored. Coordinates of plotted points are generated simply by evaluating the current equation for a series of values of the independent variable.

18

Plotting Two or More Equations

You can plot two or more equations with a single execution of DRAW or AUTO by putting the equations in a list.

To create a list of two or more equations:

1. Press (⬅)(PLOT) CAT or (➡)(ALGEBRA) to get the Equation Catalog.
2. Press (▼) and (▲) to move the pointer to an equation you want to include.
3. Press EQ+ to add the equation to the list. (If necessary, press (⬅) EQ+ to remove the last entry from the list.)
4. Repeat steps 2 and 3 for each equation you want to include.
5. Press PLOTR to begin setting up the plot for the equations.

Each time you press EQ+, a list containing the selected equations is displayed and updated in the status area. When you press PLOTR, the unnamed list is stored in *EQ*.

To plot a list of equations:

1. Set up the plot parameters using the PLOTR menu.
2. Draw the plot:
   - To use autoscaling based on the first equation, press AUTO.
   - To use the specified scaling, press DRAW.

Simply storing a list of equations in *EQ* doesn't name the list. If you don't name the list, the list is lost if you later change *EQ*. NEW adds .EQ to the name so the list is included in the Equation Catalog.

To name a list of equations currently in EQ:

1. Press (⬅)(PLOT) (➡) STEQ (the RCEQ command) to recall the list to level 1.
2. Press NEW to name the list.

To create and name a list of equations:

1. Press (⬅)(PLOT) CAT or (➡)(ALGEBRA) to get the Equation Catalog.
2. Use EQ+ to add the desired equations to the list.
3. Press →STK to copy the list onto the stack.
4. Press (ATTN) to leave the Equation Catalog.
5. Press (⬅)(PLOT) NEW to name the list.

Working in the Graphics Environment

After you execute DRAW or AUTO, the HP 48 enters the *Graphics environment*. The display shows *PICT* and the GRAPHICS menu.

The Graphics environment, like the Equation Catalog, is a special environment where the keyboard is redefined and limited to specific operations. You have access only to the GRAPHICS menu and its submenus.

To activate the Graphics environment:

- Press DRAW or AUTO in the PLOTR menu to plot and view the resulting graph.

  or
- Press (⬅)(GRAPH) (the GRAPH command) at any time.

  or
- Press (◀) if no command line is present.

To exit the Graphics environment:

- Press (ATTN).

When you exit the Graphics environment, *PICT* persists—at any time, you can press (⬅)(GRAPH) to return to the Graphics environment to view *PICT*.

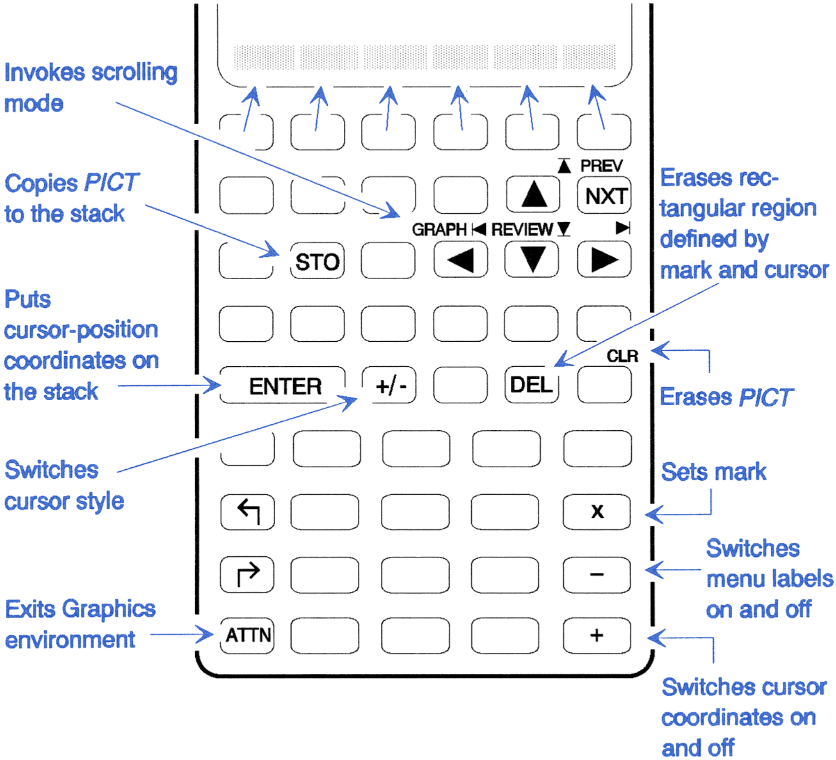
Basic Operations in the Graphics Environment

Key	Description
COORD	Displays the coordinates of the cursor position, replacing the menu keys. Press any menu key to redisplay the menu labels.
LABEL	Adds axis labels to *PICT*.
MARK	Sets the mark. If no mark exists, creates the mark at the cursor. If the mark exists at another location, moves the mark to the cursor location. If the mark exists at the cursor location, erases the mark. (All operations that require a mark create a mark at the cursor location if no mark exists.)
+/-	Switches the cursor style. In the default state (+/-), the cursor is always dark. In the alternate state (+/-■), the cursor is dark on a light background and light on a dark background.
KEYS	Erases the GRAPHICS menu keys, revealing more of the graph. Press [-] or any menu key to restore the GRAPHICS menu.
[▲] [▼] [◀] [▶]	Moves the graphics cursor in the indicated direction. When prefixed with [↱], moves the cursor to the edge of the display. If the cursor is at the edge of the display *and* if *PICT* is larger than the display, prefixing with [↱] moves the cursor to the edge of *PICT*.
[↰] [GRAPH]	Selects scrolling mode. In scrolling mode, the menu keys are erased, and, if *PICT* is larger than the display, pressing the cursor keys scrolls the display window over *PICT* in the indicated direction. Press [↰] [GRAPH] again to return to the normal Graphics environment behavior.
[ENTER]	Puts the coordinates of the cursor position on the stack.
[×]	Sets the mark (same as MARK).
[+]	Switches the cursor coordinate display on and off.
[-]	Switches the menu keys on and off.
[+/-]	Switches the cursor style (same as +/-).

Basic Operations in the Graphics Environment (continued)

Key	Description
STO	Copies *PICT* to the stack.
↰ REVIEW	Temporarily displays the PLOTR menu status message. If you hold the REVIEW key down, the status message stays until you release it.
↱ CLR	Erases *PICT*.
ATTN	Exits the Graphics environment.

18



Working with the Plot

You can do these types of operations in the Graphics environment:

- *Zoom* in or out to change the view of the plot—see the next topic below.
- Do *function analysis* to get mathematical data from the plot—see "Analyzing Functions" on page 18-25.
- Add graphical elements to the plot-see "Adding Graphical Elements to PICT" on page 19-22.)

18

Using Zoom Operations

The zoom operations in the Graphics environment let you look at a particular region of the plot in more detail (by zooming in) or look at more of the plot than is currently displayed (by zooming out).

To plot a different region without rescaling:

1. Press ▲ ▼ ◀ ▶ to move the cursor to the point you want located at the center of the display.
2. Press CENT.

To zoom by rescaling the axes:

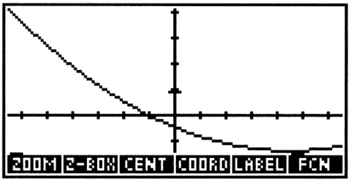
1. Press ZOOM in the GRAPHICS menu.
2. Specify the scaling:
   - To rescale the x-axis and automatically rescale the y-axis, press XAUTO, enter the x zoom factor, and press ENTER.
   - To rescale only the x-axis, press X, enter the x zoom factor, and press ENTER.
   - To rescale only the y-axis, press Y, enter the y zoom factor, and press ENTER.
   - To rescale the x- and y-axes, press XY, enter the one zoom factor used for both axes, and press ENTER.
3. Press EXIT to return to the GRAPHICS menu.

A zoom factor of 2 zooms out to show twice the axis. A zoom factor of 0.5 zooms in to show half the axis. The point at the center of the display stays at the center.

Example: Identify the number of x-axis intercepts of the expression $x^2 - 9x - 10$.

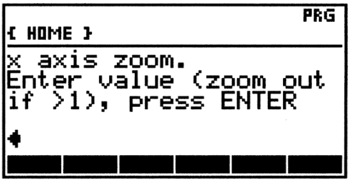
Store the expression in *EQ*, reset the plot parameters, and draw the graph using autoscaling.

['] X [y^x] 2 [−] 9 [×] X [−] 10
[↰] [PLOT] STEQ PLOTR
[NXT] RESET
[NXT] [NXT] AUTO



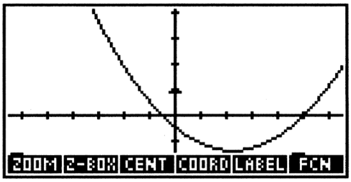
The expression has a second x-axis crossing outside the display range. Zoom out along the x-axis.

ZOOM X



Zoom out by a factor of 2. Note the second x-axis crossing.

2 [ENTER]



Press [ATTN] to return to the stack display.

To zoom in on a particular region:

1. Press [▲] [▼] [◀] [▶] to move the cursor to one corner of the desired area.
2. Press Z-BOX (or MARK or [×]) to mark the location.
3. Move the cursor:
   - To zoom in on an x-y area, move the cursor to the diagonally opposite corner of the desired area.
   - To retain the current y-axis scale, move the cursor horizontally to the other end of the x range.
   - To retain the current x-axis scale, move the cursor vertically to the other end of the y range.

4. Press Z-BOX.

To zoom in on a particular region with autoscaling:

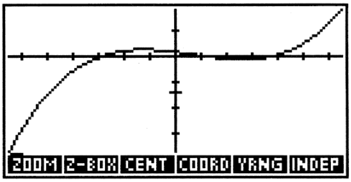
1. Press ▲ ▼ ◀ ▶ to move the cursor to one end of the desired x range (the vertical position is ignored).
2. Press Z-BOX (or MARK or ×) to mark the location.
3. Move the cursor to the other end of the x range.
4. Press ↰ Z-BOX.

The second example in the next section uses Z-BOX with autoscaling to identify the roots of the equation.

18 **Example:** Plot the earlier equation *P1* using autoscaling and the default plot parameters. Then zoom in on an area to show the behavior near the origin.

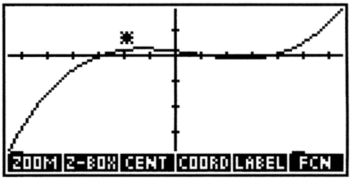
Select *P1* from the Equation Catalog and plot it.

↱ ALGEBRA (▼ if necessary)
PLOTR
NXT RESET ↰ PREV AUTO



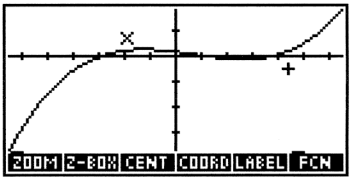
Use the cursor keys to move the cursor to the upper-left position shown below and mark the point.

▲ ◀ (as needed)
×



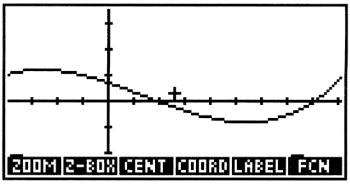
Now move the cursor to the lower-right position.

▶ ▼ (as needed)



Zoom in on the area.

Z-BOX



Press ATTN to return to the stack display.

Analyzing Functions

The GRAPHICS FCN menu lets you analyze the mathematical behavior of plotted functions. You use the graphics cursor to indicate the region or point of interest on the graph, then execute the desired calculation from the menu. You can calculate function values, slopes, areas under curves, roots, extrema and other critical points, and intersections of two curves. You can also plot derivatives of plotted functions.

To do function analysis, the current plot type must be FUNCTION. In addition, *EQ* must contain an equation, expression, or a list of equations or expressions—it can't contain a program.

To analyze a plotted function:

1. Press FCN in the GRAPHICS menu.
2. Press ▲ ▼ ◀ ▶ to move the cursor to the point you want to analyze. (For certain operations, the cursor merely needs to be *near* the point.)
3. Press the menu key for the function analysis operation you want. See the table below.
4. Press EXIT to return to the GRAPHICS menu.

When you perform a function analysis operation, the HP 48 does the following:

- Moves the cursor to the corresponding point on the function (if that point is in the display).
- Displays a message in the lower-left corner of the display showing the result.
- Returns the result to the stack as a tagged object.

The GRAPHICS FCN Menu

Key	Description
... FCN (in the GRAPHICS menu):	
ROOT	**Root.** Moves the cursor to a root (intersection of the function and the x-axis) and displays the value of the root. If the root is not in the display window, briefly displays the message OFF SCREEN before displaying the value of the root.
ISECT	**Intersection.** If only one function is plotted, moves the cursor to a root (same as ROOT). If two or more functions are plotted, moves the cursor to the closest intersection of two functions and displays the (x,y) coordinates. If the closest intersection is not in the display window, briefly displays the message OFF SCREEN before displaying the coordinates of the intersection.
SLOPE	**Slope.** Calculates and displays the slope of the function at the x-value of the cursor, and moves the cursor to the point on the function where the slope was calculated.
AREA	**Area.** Calculates and displays the area beneath the curve between two x-values defined by the mark and cursor. (Before you execute this operation, press ⊠ to mark one end of the x interval, then move the cursor to the other end.)
EXTR	**Extremum.** Moves the cursor to an extremum (local minimum or maximum) or other critical point and displays the (x,y) coordinates. If the closest extremum or inflection point is not in the display window, briefly displays the message OFF SCREEN before displaying the value.
EXIT	**Exit.** Exits the GRAPHICS FCN menu back to the main GRAPHICS menu.

The GRAPHICS FCN Menu (continued)

Key	Description
F(X)	**Function Value.** Displays the function value at the current x-value of the cursor, and moves the cursor to that point on the function curve.
F'	**Derivative Plot.** Plots the first derivative of the function and replots the original function. Also adds the symbolic expression for the first derivative to the contents of *EQ*. (If *EQ* is a list, F' adds the expression to the front of the list. If *EQ* is not a list, F' creates a list and inserts the expression to the front of the list.)
NXEQ	**Next Equation.** Rotates the list in *EQ* and displays the equation now at the beginning of the list. (The second equation is moved to the beginning of the list and the first equation is moved to the end.)

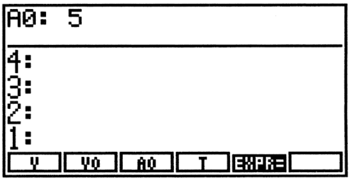
If you've plotted two or more equations by storing a list in *EQ* (see "Plotting Two or More Equations" on page 18-18), the function analysis operations use the first equation in the list, unless otherwise stated. Press NXEQ in the FCN menu to rotate equations within the list.

Example: An equation for velocity at constant acceleration is $v = v_0 + a_0 t$.

For an initial velocity $v_0 = 10$, and a constant acceleration $a_0 = 5$, find the velocity at $t = 2$ and find the total displacement x between $t = 0$ and $t = 10$. (The displacement is the area under the curve of velocity vs. time.)

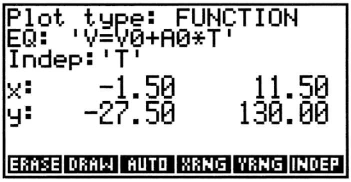
Key in the equation and store it in *EQ* without naming it. Use the SOLVE menu because you can easily store values for v_0 and a_0 using the SOLVR menu.

['] V [←][=] V0 [+] A0 [×] T
[←][SOLVE] STEQ
SOLVR 10 V0 5 A0



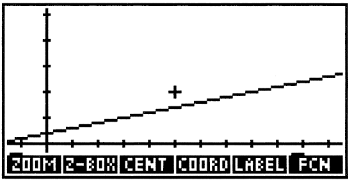
Set the display mode to 2 Fix so that coordinates and function analysis results are easy to read in the Graphics environment. Then get the PLOTR menu. To obtain integer values for the x- and y-axis tick marks, use SCALE to specify 1 unit per x-axis tick mark and 25 units per y-axis tick mark. This enables exact calculations. Use CENT to specify the plot center at (5,50). Finally, specify T as the independent variable.

↰ MODES 2 FIX
↱ PLOT NXT
1 SPC 25 SCALE
↰ ( ) 5 SPC 50 CENT
18 ↰ PREV ' T INDEP



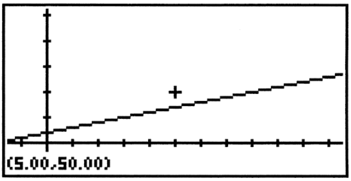
Erase *PICT*, then draw the graph.

ERASE DRAW



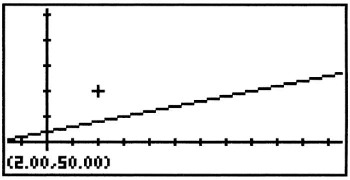
Check the coordinates of the graphics cursor. The x-coordinate (the value of T) is 5.

COORD (or +)



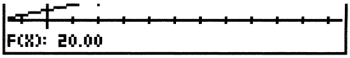
Hold down ◀ until the displayed x-coordinate is exactly 2.00. (The cursor moves slowly when coordinates are displayed.)

◀ (hold down)



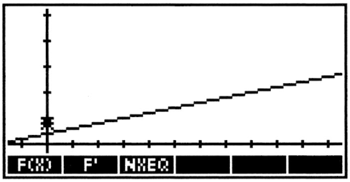
Press any menu key (or [+] or [−]) to redisplay the menu labels. Then find the value of the function at $T = 2$. The velocity is 20.

[+]
FCN [NXT] F(X)



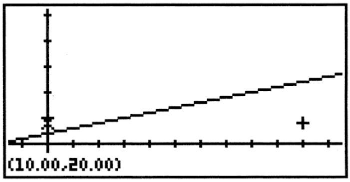
Now calculate the displacement between $T = 0$ and $T = 10$. First, restore the menu keys. Then move the cursor to the y-axis ($T = 0$) and set the mark.

[+]
hold down [◀]
[×]



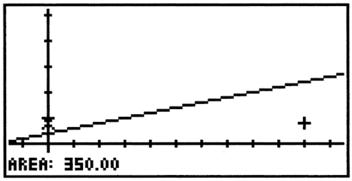
Display the cursor coordinates, move the cursor to the right edge of the display, then back until its x-coordinate is 10.

[+]
[↱][▶]
then hold down [◀]



Redisplay the menu labels, and calculate the area—the displacement.

[+] [NXT] AREA



Return to the stack and note that the function value and area have been returned to the stack as tagged objects.

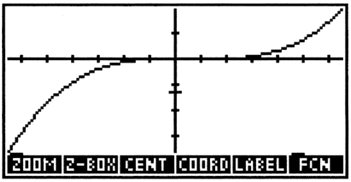
[ATTN] [ATTN]

Example: For the expression $x^3 - 2x^2 - x + 2$ find the following:

- The number of real roots.
- The value of the leftmost root.
- The slope of the expression at the leftmost root.
- The value of the expression at the y-axis ($x = 0$).
- The coordinates of the local minimum.

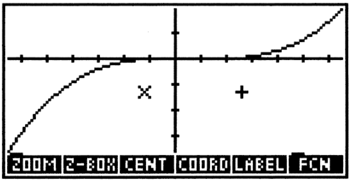
Key in the expression and store it in EQ. Reset the plot parameters, then draw the graph using autoscaling for the y-axis.

['] X [y^x] 3 [−] 2 [×] X [y^x] 2
[−] X [+] 2
18 [⬅] [PLOT] STEQ
PLOTR [NXT] RESET
[⬅] [PREV] AUTO



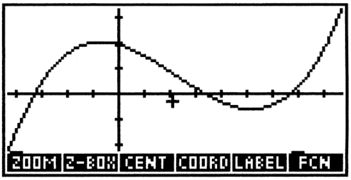
The region of interest needs enlargement, so set the mark and cursor as shown.

[◀] (holds down)
[×]
[▶] (hold down)



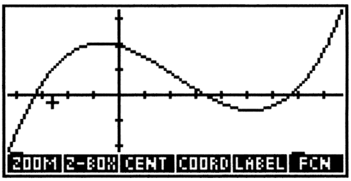
Now zoom-to-box, autoscaling the y-axis. You can now see that there are three real roots in this region.

[⬅] Z-BOX



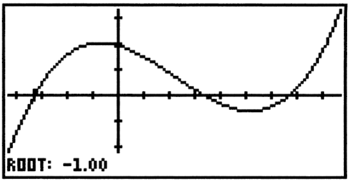
Move the cursor near the leftmost root.

[◀] (hold down)



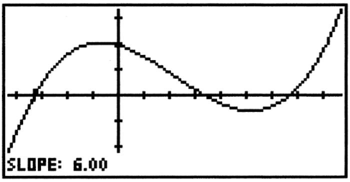
Find the value of the root. The cursor moves to the root and the value of the root is displayed in the lower left corner.

FCN ROOT



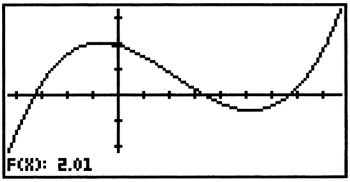
Calculate the slope of the function at the root. (Press any key to redisplay the menu labels.) The value you obtain for the slope may vary slightly from that shown in the following display, depending on the exact coordinates of the rectangular region you defined with (⬅) Z-BOX.

(−) SLOPE



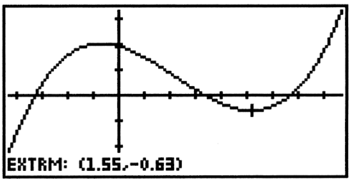
Move the cursor to the y-axis ($x = 0$) and find the value of the function. The cursor moves to the corresponding point on the function.

(+)
(▶) (hold down)
(NXT) F(X) .



Move the cursor to an x-axis value near the minimum and find the coordinates of the local minimum.

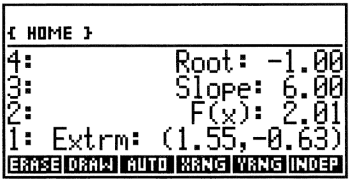
(+)
(▶) (hold down)
(NXT) EXTR



18

Leave the Graphics environment and note that the results have been returned to the stack as tagged objects.

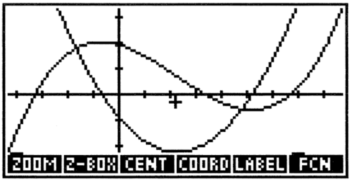
[ATTN] [ATTN]



Example: For the expression in the previous example, plot the derivative of the expression and find the coordinates of the positive x value where the derivative and the original expression are equal.

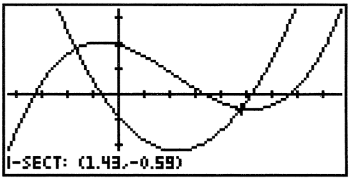
Return to the Graphics environment and plot the derivative.

[←][GRAPH] FCN [NXT] F'



Move the cursor near the positive intersection and find the intersection.

[▶] (hold down)
FCN ISECT



Press [ATTN] [ATTN] [←][MODES] STD to return to the stack display and Standard display mode.

More about Function Analysis

Analyzing Difficult Plots

Each of the previous function analysis examples has generated a plot in which the intersection of the x- and y-axes is visible in the display, providing you with immediate orientation. However, depending on the expression and the current display ranges, one or both axes may not be visible. In such cases, you can press ⬅ REVIEW to determine what part of the graph you're looking at.

For example, suppose you plot a graph with autoscaling and don't have an x-axis in your graph. If you press ⬅ REVIEW and see a y-axis display range from 230 to 410, you know the portion of the graph you're currently viewing is above the x-axis.

You can use the following ideas to analyze such functions:

- If you want to better understand the general shape of the function and its relationship to the axes, you can zoom out to see more of the function. XAUTO is particularly suitable for such exploratory zooming.
- If you want to identify a particular feature of the function, such as a root or extremum, you can execute the corresponding operation in the GRAPHICS FCN menu to return the coordinates of that feature to the stack. Then leave the Graphics environment and use CENT from the PLOTR menu to bring the feature into view when you redraw the graph. Analysis of the function's shape at the feature may provide insight into the relative position of other features on the curve. Subsequent zoom operations may then be appropriate.

18

How the Function Analysis Operations Work

The operations in the GRAPHICS FCN menu are linked to commands that you can execute outside the Graphics environment. (In this list, the normal distinction between expressions and equations is reasserted.)

ROOT — Executes ROOT (the numeric root-finder in the HP Solve application) to find an x-axis intersection. If there are multiple roots (intersections), the root-finder usually finds the root closest to the current cursor location. For an equation, it searches for a root of the expression on the right side of the equation.

ISECT — Executes ROOT. For a single expression or for an equation whose left side has not been plotted (flag −30 clear), ISECT works just like ROOT. For an equation whose left and right sides have been plotted (flag −30 set), it finds the nearest intersection of the left and right sides. For two expressions, it finds the nearest intersection of the expressions. For two equations, it finds the nearest intersection of the right sides.

SLOPE Executes ∂, then evaluates the resultant expression at the x-value of the cursor.

AREA Executes $\int$, using the x-values defined by the mark and cursor as limits.

EXTR Executes ∂, then finds the x-value closest to the cursor that causes the resultant expression to evaluate to zero.

F(X) Evaluates the expression at the x-value defined by the cursor.

18

F' Executes ∂, then puts the resultant expression in a list in EQ with the original expression, and plots the list.

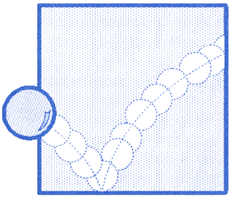
Summary of Zoom and Function Analysis Operations

Zoom and Function Operations in the Graphics Environment

Key	Description
ZOOM	Displays the GRAPHICS ZOOM menu, which allows you to rescale and recenter the plot. (See "Using Zoom Operations" on page 18-22.)
Z-BOX	Redraws the graph so the rectangular area whose opposite corners are defined by the mark and cursor fills the display. (⬅ Z-BOX redraws the graph so that the x-range defined by the mark and cursor fills the display, and *autoscales* the y-axis.)
CENT	Redraws the graph with the current cursor position at the center of the display.
COORD	Displays the coordinates of the cursor position, replacing the menu keys. Press any menu key to redisplay the menu labels.
FCN	Displays the GRAPHICS FCN menu for analyzing function plots. (See "Analyzing Functions" on page 18-25.)

19

More about Plotting and Graphics Objects



The previous chapter covered basic plotting of mathematical functions: the plot type was specified as FUNCTION in all examples, and a limited set of plot parameters was covered. This chapter extends the concepts introduced in chapter 18:

- Specifying special options for plots.
- Working with plot coordinates.
- Changing the size of *PICT*.
- Drawing conic, polar, parametric, truth, and statistical plots.
- Plotting programs and user-defined functions.
- Plotting with units.
- Adding graphical elements to *PICT*.
- Working with graphics objects on the stack.

Refining Plots

You can refine your plots by changing the standard plot setup:

- Plotting a specified part of a display range.
- Specifying special axes labels.
- Specifying a different sampling frequency.

Using Plotting Range instead of Display Range

The *plotting range* is the range of the independent variable over which the current equation is evaluated. If you don't specify the plotting range, the HP 48 uses the x-axis display range (specified by XRNG) as the plotting range. However, you can specify a plotting range that's different from the x-axis display range:

- For polar and parametric plots, the independent variable isn't related to the x-axis variable—so you specify the plotting range to control the range of the independent variable.
- For truth and conic plots, you can shorten plotting time by specifying plotting ranges that are shorter than the x- and y-axis display ranges. These plot types require you to specify the *dependent* variable—you can specify its plotting range different from the y-axis display range.

To specify a plotting range for a variable:

19

1. Enter the plotting range:
   - To specify only the range, enter the two limits for the range (press SPC or ENTER to separate the numbers).
   - To specify the variable name *and* its plotting range, enter a list (with { } delimiters) containing the variable name and the two limits for the range, and press ENTER.
2. Set the plotting range:
   - To set the plotting range for the independent variable or x-axis variable, press INDEP in the PLOTR menu.
   - To set the plotting range for the dependent variable, press DEPN in the PLOTR menu.

If you use INDEP and DEPND with a list to specify both the independent or dependent variable and its plotting range, the list has the form

{ *name lower upper* }

Example: Specify the plotting range of the independent variable to be 0 through +10. Press 0 SPC 10 INDEP.

Example: Specify the independent variable to be T and its plotting range to be 0 through +10. Press ← { } T ▶ 0 SPC 10 ENTER INDEP.

Specifying Axes and Labels

If the axes are in the plotting range, AUTO and DRAW automatically draw them with tick marks placed at 10-pixel intervals. The axes normally intersect at (0,0).

You can change the coordinates of the intersection point. You can label the axes with their names and their extreme numeric values. You can also specify axis labels that are different from the independent and dependent variables names.

To specify the intersection point or labels for the axes:

1. Enter the information:
   - To specify the intersection point, enter the (x,y) coordinates as a complex number.
   - To specify axis labels, enter a list (with { } delimiters) containing the string for the x-axis label and the string for the y-axis label, and press ENTER.
2. Press AXES in the PLOTR menu.

19

You can specify both the intersection point and labels using a list of the form

{ (x, y) "*x-label*" "*y-label*" }

To label the axes using the current AXES data:

- Press LABEL in the PLOTR menu.

LABEL displays in *PICT* the names of the independent and dependent variables (unless you've specified labels) and the coordinates of the end-points of the axes (using the current display format).

Example: Assign the label X2 to the horizontal axis and the label F(X2) to the vertical axis (regardless of the names of the independent and dependent variables), then label the axes. (The AXES list is { "X2" "F(X2)" }.) Press ◄ { } ► " " X2 ▶ ► " " F(X2) ENTER AXES LABEL.

Specifying Resolution

You can specify the interval between values of the independent variable used to generate the plot. A larger resolution gives faster plots, but decreases the accuracy of the line connecting the points.

To change the resolution:

- To enter user units, enter a number for the resolution, then press RES in the PLOTR menu.
- To enter pixels, enter a binary integer (with # delimiter) for the number of pixels, then press RES in the PLOTR menu.
- To restore the default resolution, enter 0 (or # 0), then press RES in the PLOTR menu.

19

For all plot types, RES uses a real number argument to define the interval in user units. For FUNCTION, CONIC, and TRUTH plot types, you can specify the interval in pixels using a binary integer argument. (For POLAR and PARAMETRIC plot types, a binary integer argument doesn't apply.) The resolution interval is also used for making certain statistical plots. The default intervals for different plot types are listed below.

Default Resolution Intervals

Plot Type	Default Interval
Equation:	
FUNCTION	1 pixel (point plotted in every pixel column)
CONIC	1 pixel (point plotted in every pixel column)
TRUTH	1 pixel (point plotted in every pixel column)
POLAR	2°, 2 grads, or $\pi/90$ radians
PARAMETRIC	(independent variable range in user units)/130
Statistical Data:	
BAR	10 pixels (specifies bar width)
HISTOGRAM	10 pixels (specifies bar width)
SCATTER	(not applicable)

Summary of Plot-Refinement PLOTR Menu Operations

The following commands in the PLOTR menu let you tailor plot features.

The PLOTR Menu—Plot Refinement Operations

Key	Programmable Command	Description
(⮌)(PLOT) PLOTR or (⮎)(ALGEBRA) PLOTR:		
INDEP	INDEP	Sets the name in level 1 as the independent variable. INDEP can also specify the *plotting* range for the independent variable. ((⮎) INDEP recalls the current independent variable, and its plotting range if specified.)
DEPN	DEPND	Sets the name in level 1 as the dependent variable (for conic and truth plots). DEPND can also specify the plotting range for the dependent variable. ((⮎) DEPN recalls the current dependent variable, and its plotting range if specified.)
RES	RES	Sets the plot *resolution*. ((⮎) RES recalls the current resolution.)
AXES	AXES	Sets the coordinates of the axes intersection using the complex-number argument from level 1. AXES can also specify axes labels that are different from INDEP and DEPND. ((⮎) AXES returns the current axes intersection.)

19

The PLOTR Menu—Plot Refinement Operations (continued)

Key	Programmable Command	Description
DRAX	DRAX	Adds axes to *PICT*. (Not necessary if you execute DRAW or AUTO from the keyboard.)
LABEL	LABEL	Adds axes labels to *PICT*.
*H	*H	Multiplies the vertical scale by the level 1 argument *n*. (Zooms *in* if $n<1$.)
*W	*W	Multiplies the horizontal scale by the level 1 argument *n*. (Zooms *in* if $n<1$.)
PDIM	PDIM	Changes the size of *PICT*. (⮫ PDIM returns the size of *PICT*.)
⮪ REVIEW		Redisplays the plot parameters.

19

Understanding the PPAR Variable

The HP 48 uses a built-in plot parameter variable named *PPAR* to store the plotting parameters. You normally control the plot parameters using commands in the PLOTR menu. Because *PPAR* is a variable, you can have a different *PPAR* in every directory. *PPAR* contains a list with the following objects:

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

Contents of the PPAR List

Element	Description	Default
$(x_{\text{min}}, y_{\text{min}})$	A complex number representing the coordinates of the lower left corner of the display range.	(−6.5,−3.1)
$(x_{\text{max}}, y_{\text{max}})$	A complex number representing the coordinates of the upper right corner of the display range.	(6.5,3.2)
indep	Independent variable. The name of the variable, or a list containing the name and two real numbers (the horizontal plotting range).	*X*
res	Resolution. For equations, a real number or binary integer representing the interval between plotted points. For statistical data, the meaning varies.	0 (points plotted in every pixel column)
axes	A complex number representing the coordinates of the axes intersection, or a list containing the intersection and labels (strings) for both axes.	(0,0)
ptype	Command name specifying the plot type.	FUNCTION
depend	Dependent variable. The name of the variable, or a list containing the name and two real numbers (the vertical plotting range).	*Y*

19

To reset PPAR to its default:

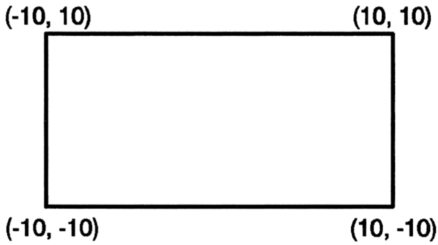
- Press RESET in the PLOTR menu.

The RESET operation resets all parameters in *PPAR* to their default states—except the plot type—and erases *PICT* and restores it to its default size.

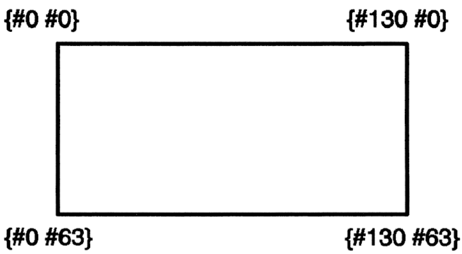
Using Plot Coordinates

The size of *PICT* (or any graphics object on the stack), or the position of a point within it, are expressed in terms of horizontal and vertical coordinates. There are two unit systems for plot coordinates:

- **User-unit coordinates.** (Or simply "units".) Represented by a complex number, giving the horizontal and vertical coordinates. They're interpreted according to the first two parameters in *PPAR*, (x_{min}, y_{min}) and (x_{max}, y_{max}). For example, if (x_{min}, y_{min}) is (-10, -10) and (x_{max}, y_{max}) is (10, 10), coordinates (-10, 10) represent the upper-left pixel in the graphics object. (Graphics objects on the stack don't have user-unit coordinates.)



- **Pixel coordinates.** Represented by a list containing two binary integers, the horizontal and the vertical pixel numbers. For example, {#0 #0} represents the upper-left pixel.



To convert a coordinate to the other type:

- To convert user-units to pixels, enter the complex number (x, y) and press PRG DSPL NXT C→PX.
- To convert pixels to user-units, enter the list { $\#n_x$ $\#n_y$ } and press PRG DSPL NXT PX→C.

The conversion uses the current parameters in *PPAR*.

Coordinate Conversion Commands

Key	Programmable Command	Description
PRG DSPL (page 2):		
PX→C	PX→C	Converts pixel coordinates to user-unit coordinates. Takes the list argument { $\#n_x$ $\#n_y$ } from level 1 and returns (x, y).
C→PX	C→PX	Converts user-unit coordinates to pixel coordinates. Takes (x, y) from level 1 and returns { $\#n_x$ $\#n_y$ }.

19

Changing the Size of PICT

You can make *PICT* larger than its default size (131 by 64 pixels)—and either keep the same x and y scale factors extended over the new size, or keep the same x and y display ranges over the new size.

To change the size of PICT:

- To keep the same scaling, enter two complex numbers (with ( ) delimiters) specifying the coordinates of diagonally opposite corners in user-units, then press PDIM in the PLOTR menu.
- To keep the same display ranges, enter two binary integers (with # delimiter) specifying the horizontal and vertical sizes in pixels, then press PDIM in the PLOTR menu.

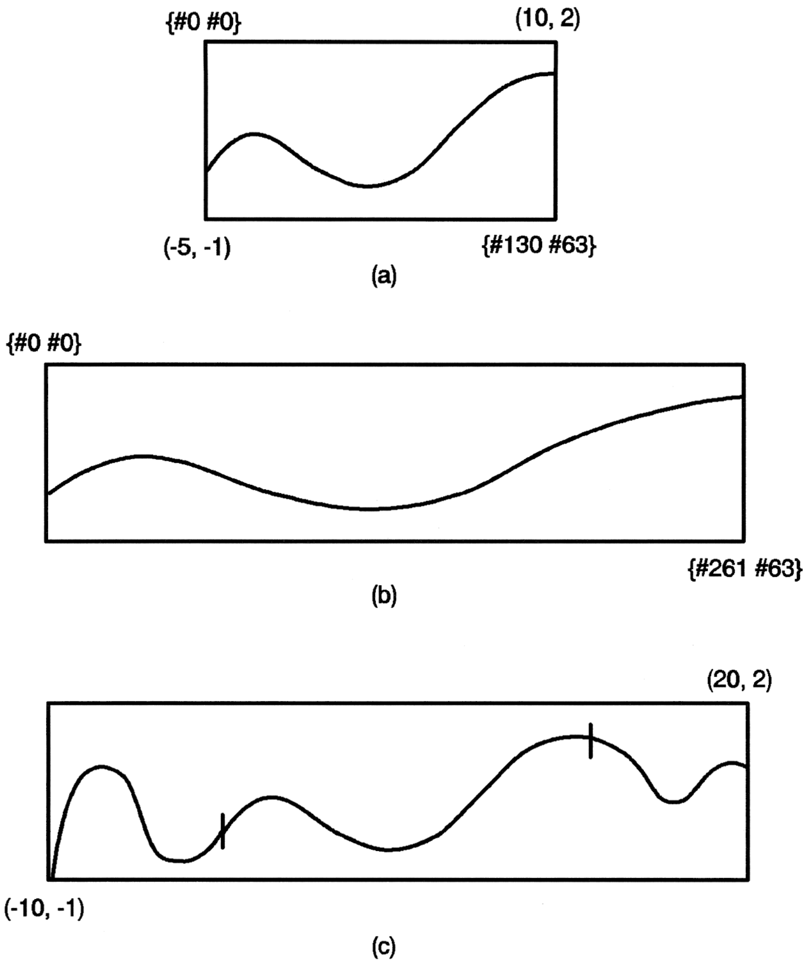
The result of the PDIM (*PICT dimension*) command depends on the type of coordinates—user-units or pixels—though both forms change the size of *PICT*.

Example: Suppose *PICT* is currently its default size (#131 wide by #64 high in pixel units), and the current x-axis display range is -5 to 10 and the y-axis display range is -1 to 2. Assume *PICT* contains the graph shown in figure (a) below.

To double the x range of *PICT* in the horizontal direction and keep the same scales (units per pixel), enter `(-10,-1)` and `(20,2)` and press `PDIM`. (*PICT* becomes #262 wide by #64 high in pixel units.) If you redraw the graph, the effect is to add more points to the graph at both ends, shown in figure (c).

19

To double the size of *PICT* in the horizontal direction and the keep the display ranges the same, enter `#262` and `#64` and press `PDIM`. (The scale of the x-axis in units per pixel is halved.) If you redraw the graph, the effect is to "stretch" the graph, shown in figure (b).



Changing the Size of *PICT*

Choosing Plot Types

The plot type tells the HP 48 how to interpret the current equation (or the current statistical data for statistical plot types). The PLOT menu status message indicates the current plot type.

You can choose eight different plot types:

- **Equations.** FUNCTION, CONIC, POLAR, PARAMETRIC, and TRUTH.
- **Statistical data.** SCATTER, HISTOGRAM, and BAR. (See "Drawing Statistical Plots" on page 19-21.)

To check the current plot type:

- Press ⬅ PLOT.

To change the plot type:

1. Press PTYPE in the PLOT or PLOTR menu.
2. Press a menu key to select one of the eight plot types.

The PTYPE Menu

Key	Programmable Command	Description
... PTYPE (in PLOTR menu):		
		Equation.
FUNC	FUNCTION	Prepares to plot equations that return a unique $f(x)$ for each value of x.
CONIC	CONIC	Prepares to plot conic sections—circles, ellipses, parabolas, and hyperbolas.
POLAR	POLAR	Prepares to plot expressions that return a radius for each value of the specified polar angle.
PARA	PARAMETRIC	Prepares to plot equations that return a complex result for each value of the specified independent variable.
TRUTH	TRUTH	Prepares to plot expressions that return a true (1) or false (0) value for each pair of x and y values, such as equations with comparison functions.
		Statistical Data.
BAR	BAR	Prepares to draw a bar chart of the data from a specified column (XCOL) of the statistical matrix.
HIST	HISTOGRAM	Prepares to draw a frequency histogram of the data from a specified column (YCOL) of the statistical matrix.
SCATT	SCATTER	Prepares to plot points from two columns (XCOL and YCOL) of the statistical matrix.

Function Plots

FUNCTION is the default plot type. All the examples in chapter 18 used the FUNCTION plot type.

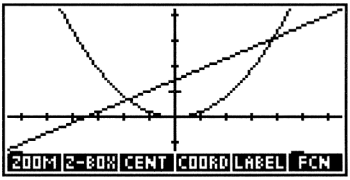
The FUNCTION Plot Type

Form of Current Equation	Example	Points Plotted
$f(x)$	$x^3 - 5x^2 + 20$	$(x, f(x))$
$y = f(x)$	$y = x^2 + x + 4$	Flag -30 clear: $(x, f(x))$ Flag -30 set: $(x, f(x))$ and (x, y)
$f(x) = g(x)$	$x^2 = 2x + 7$	$(x, f(x))$ and $(x, g(x))$

Example: Plot the equation $x^2 = 2x + 7$.

Set the plot type to FUNCTION, reset the plotting parameters, and plot an autoscaled graph. (The x-values at which the two lines intersect are roots of the equation.)

(←) (PLOT)
(') X (y^x) 2
(←) (=) 2 (×) X (+) 7 STEQ
PTYPE FUNC
PLOTR (NXT) RESET
(NXT) (NXT) AUTO



Press (ATTN) to return to the stack display.

Conic Sections

The equation for a conic section is second degree or less in both x and y. For example, the following equations are all valid equations for plotting conic sections:

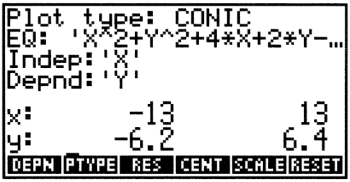
$$x^2 + y^2 + 4x + 2y - 5 = 0 \quad \text{(circle)}$$
$$5x^2 + 3y^2 - 18 = 0 \quad \text{(ellipse)}$$
$$x^2 - 4x + 3y + 2 = 0 \quad \text{(parabola)}$$
$$2x^2 - 3y^2 + 3y - 5 = 0 \quad \text{(hyperbola)}$$

Note that the variable specified by DEPND *is* used when the plot type is CONIC. Also note that autoscaling may not be useful for conic sections—you can use CENT and SCALE instead.

Example: Plot the conic section for the equation $x^2 + y^2 + 4x + 2y - 5 = 0$.

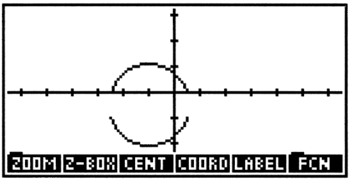
Set the plot type to CONIC, set the plot parameters, and use CENT and SCALE to draw a "round" circle. 19

[←] [PLOT]
['] X [y^x] 2 [+]
Y [y^x] 2 [+] 4 [×] X
[+] 2 [×] Y [−] 5 [←] [=] 0
STEQ PTYPE CONIC
PLOTR ['] X INDEP
[NXT] ['] Y DEPN
[←] [( )] 0 [SPC] 0 CENT
2 [SPC] 2 SCALE



Plot the conic section.

[←] [PREV]
ERASE DRAW

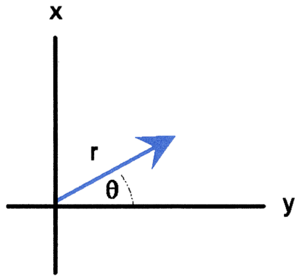


Press [ATTN] to return to the stack display.

For conic plots, the HP 48 actually plots the two branches of the conic section separately. This may introduce one or two discontinuities in the connected graph, as in the previous example. Specifying a finer resolution (decreasing the interval between plotted points) helps eliminate discontinuities (see "Specifying Resolution" on page 19-3).

Polar Plots

In polar plots, the polar angle is the independent variable—θ in this illustration.



The POLAR Plot Type

Form of Current Equation	Example	Points Plotted
$f(\theta)$	$\cos\theta + \sin\theta$	$(f(\theta), \angle\theta)$
$r = f(\theta)$	$r = 2\cos\theta$	$(f(\theta), \angle\theta)$
$\theta = constant$	$\theta = 0.2\pi$	radial line
$f(\theta) = g(\theta)$	$4\sin\theta = r^2$	$(f(\theta), \angle\theta)$ and $(g(\theta), \angle\theta)$

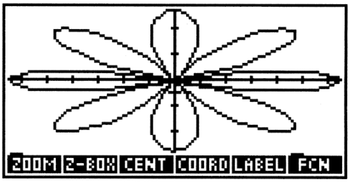
Unless you specify otherwise, the plots are drawn for a full circle of the independent variable θ (0 through 360 degrees, 2π radians, or 400 grads, according to the current angle mode). See "Using Plotting Range instead of Display Range" on page 19-1.

If you use autoscaling, the HP 48 computes an appropriate x- and y-axis display range based on the θ-range—but the resulting x- and y-axis scales may differ.

Example: Plot the polar equation $r = 2\cos(4\theta)$ for values of θ in the default range 0° through 360°. (This example assumes Degrees mode is active.)

Store the equation in *POL*. (To key in θ, press [α] [→] F). Select POLAR plot type, specify the independent variable θ, then draw the plot using autoscaling.

[←] [RAD] (if necessary)
[←] [PLOT]
['] R [←] [=] 2 [×] [COS] 4 [×] θ
NEW POL [ENTER]
PTYPE POLAR
PLOTR ['] θ INDEP
AUTO



19

Press [ATTN] to return to the stack display.

In this example, autoscaling generates different x- and y-axis scales, compressing the plot in the vertical direction.

Parametric Plots

In parametric equations, two dependent variables (typically x and y), represented by the horizontal and vertical axes, are expressed as functions of an independent variable (typically t).

For example, these parametric equations define x and y in terms of the independent variable t:

$$x = t^2 - t \qquad \text{and} \qquad y = t^3 - 3t$$

To plot a parametric equation, the equation or program must return a complex result giving the coordinates (x,y). You must also specify the plotting range for the independent variable—it's unrelated to the x-axis display range. See "Using Plotting Range instead of Display Range" on page 19-1.

To plot the equations shown above, you can write them as an expression that returns the complex result $x + yi$:

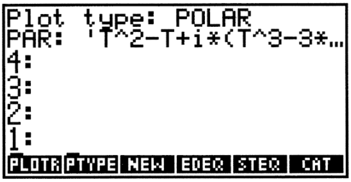
```
'T^2-T+i*(T^3-3*T)'
```

If you use autoscaling, the HP 48 computes an appropriate x- and y-axis display range based on the plotting range of the independent variable.

Example: Plot the equations shown above for values of t in the range -3 through $+3$.

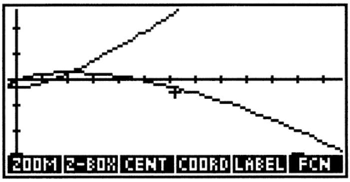
Store the expression shown above in *PAR*. (To key in the complex number i, press [α] [⬅] I.)

['] T [y^x] 2 [−] T [+]
i [×] [⬅] [( )] T [y^x] 3
[−] 3 [×] T
[⬅] [PLOT] NEW
PAR [ENTER]



Set the plot type to PARAMETRIC, specify the independent variable and its plotting range, and draw the graph using autoscaling.

PTYPE PARA
PLOTR [⬅] [{ }] T [SPC] 3 [+/−] [SPC]
3 [ENTER] INDEP
AUTO



Press [ATTN] to return to the stack display.

Truth (Relational) Plots

Truth plots evaluate expressions that return true (any nonzero real number) or false (0) results. At the coordinates for each pixel, the pixel is turned *on* if the expression is true—it's *unchanged* if the expression is false.

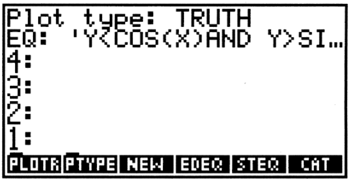
The variable you specify using DEPND defines the (independent) variable for the vertical axis.

Unless otherwise specified, every pixel in the display is evaluated. You can speed up the plot by specifying a smaller x and y plotting range. See "Using Plotting Range instead of Display Range" on page 19-1.

Example: Draw a truth plot for the expression 'Y<COS(X) AND Y>SIN(X)' over the x-axis display range $-\pi$ to $\pi/2$ radians and y-axis display range of -1.5 to 1.5. To shorten the plotting time, specify smaller plotting ranges.

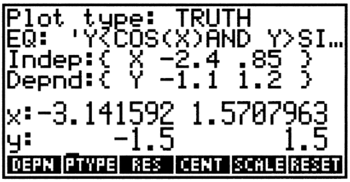
Select Radians mode, and store the expression in EQ. (To type ≤, press (α) (←) 2. To type ≥, press (α) (→) 2.) Set the plot type to TRUTH.

(←)(RAD) (if necessary)
(') Y < (COS) X (▶)
(PRG) TEST AND
Y > (SIN) X
(←)(PLOT) STEQ
PTYPE TRUTH



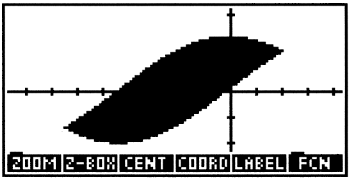
Specify the display ranges—use −1.5 to 1.5 for the y-axis. Specify the horizontal and vertical variables, and limit their plotting ranges: −2.4 to .85 for X, and −1.1 to 1.2 for Y.

PLOTR
(←) (π) (+/−) (→)(→NUM)
(ENTER) 2 (÷) (+/−) XRNG
1.5 (+/−) (SPC) 1.5 YRNG
(←)({}) X (SPC) 2.4 (+/−) (SPC) .85
(ENTER) INDEP
(←)({}) Y (SPC) 1.1 (+/−) (SPC) 1.2
(ENTER) (NXT) DEPN



Draw the plot. (This takes several minutes.)

(←)(PREV) ERASE DRAW



Press (ATTN) (←)(RAD) to return to the stack display and Degrees mode.

Plotting Programs and User-Defined Functions

You can plot more than just expressions and equations—you can also plot programs. And your expressions, equations, and programs can include user-defined functions.

You can plot a program if it takes nothing from the stack, uses the independent variable in the program, and returns exactly one untagged number to the stack:

- **Real result.** Equivalent to the expressions $f(x)$ (type FUNCTION) and $r(\theta)$ (type POLAR). For example, the program

19

```
« IF 'X<10' THEN '3*X^3-45*X^2+350' ELSE 1000 END»
```

  plots

$$f(x) = \begin{cases} 3x^3 - 45x^2 + 350 & \text{if } x < 10 \\ 1000 & \text{if } x \geq 10 \end{cases}$$

- **Complex result.** Equivalent to $(x(t), y(t))$ (type PARAMETRIC). For example, the program

```
« 't^2-2' →NUM 't^3-2t+1' →NUM R→C »
```

  plots the parametric equations

$$x = t^2 - 2 \qquad \text{and} \qquad y = t^3 - 2t + 1$$

To plot a program, store the program or its name in *EQ*. Note that you can't use the operations in the GRAPHICS FCN menu with plots of programs.

To plot a user-defined function, include it in an expression, equation, or program. For example, if you've created the *COT* (cotangent) user-defined function, you can plot the expression `'COT(X)'`, where X is the independent variable.

Plotting with Units

You can plot equations that contain unit objects if you observe these restrictions:

- If the independent variable requires units for *EQ* to evaluate properly, you must store a unit object in the independent variable *before* plotting. (The number part of the unit object you store is ignored during plotting.)
- If evaluation of *EQ* returns a unit object, only the scalar part of the unit object is used for plotting.

Note that *no* automatic conversions are performed on the plotted values. If the desired units for the x- or y-axis is m (meters), values are *not* converted to meters if the value has units of ft (feet). 19

Drawing Statistical Plots

You can use two different applications to plot statistical data (data you've accumulated in the statistics variable ΣDAT):

- **Statistics application.** This is the simplest way to plot statistical data. It's explained in "Plotting Statistical Data" on page 21-13.
- **Plot application.** This lets you control more of the plot parameters. It's explained below.

Plots of statistical data are similar to plots of mathematical data, except that:

- The data comes from the reserved variable ΣDAT, rather than from *EQ*.
- Instead of specifying independent and dependent variables in *PPAR*, you specify analogous *columns* of statistical data in the reserved variable ΣPAR.
- The plot type is specified as BAR, HISTOGRAM, or SCATTER.

To plot statistical data from the Plot application:

1. Change the plot type to BAR, HISTOGRAM, or SCATTER.

2. Specify appropriate plot parameters. (Press (→)(STAT) to find the XCOL and YCOL commands for specifying the x and y columns of ΣDAT.)
3. Press DRAW or AUTO to plot the graph.

When you specify a statistical plot type in the Plot application:

- The status message in the PLOT menu changes to show you the contents of ΣDAT, rather than EQ. The plot data comes from ΣDAT.
- The status message in the PLOTR menu changes to show you the contents of ΣDAT, the columns in ΣDAT corresponding to the x- and y-axes, and the currently specified statistical model.
- The independent and dependent variables correspond to the *column* numbers specified in ΣPAR, rather than variable names specified in $PPAR$.

19

The Plot application lets you specify plot parameters for statistical plots that aren't available to you in the Statistics application, such as

- RES lets you specify the number of bins in a histogram plot.
- CENTR and SCALE let you specify, for a scatter plot, display ranges that are larger than the range of plotted points.
- AXES lets you specify labels for the axes in a bar plot.

Adding Graphical Elements to PICT

You can add graphical elements to $PICT$ using interactive operations in the Graphics environment and using commands.

Adding Elements Using the Graphics Environment

To add graphical elements interactively:

1. View the Graphics environment.
2. Use the GRAPHICS menu to add the element:
   - For elements that use a mark, move the cursor to the first point, press (×) or MARK to mark it, move the cursor to the second point, then press the menu key for the operation.

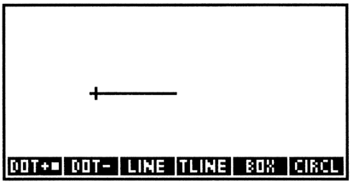
■ For other elements, press the menu key for the operation.

Graphical Element Operations in the Graphics Environment

Key	Description
DOT+	Turns line-drawing on and off. While turned on, pixels beneath the cursor are turned on as you move the cursor across the display. While line-drawing is active, DOT+■ is displayed.
DOT-	Turns line-erase on and off. While turned on, pixels beneath the cursor are turned off as you move the cursor across the display. While line-erase is active, DOT-■ is displayed.
LINE	Draws a line between the mark and the cursor, and moves the mark to the cursor.
TLINE	Toggles pixels on and off along a line between the mark and cursor. Does not move the mark to the cursor.
BOX	Draws a rectangular box using the mark and cursor as opposite corners.
CIRCL	Draws circle centered at the mark with radius defined by the mark and cursor.
MARK	Sets the mark. If no mark exists, creates the mark at the cursor. If the mark exists at another location, moves the mark to the cursor location. If the mark exists at the cursor location, erases the mark. (All operations requiring a mark create a mark at the cursor location if no mark exists.)
DEL	Erases the rectangular region whose opposite corners are defined by the mark and cursor.
⮫ CLR	Clears *PICT*.
×	Sets the mark (same as MARK).
DEL	Erases a rectangle, same as DEL .

19

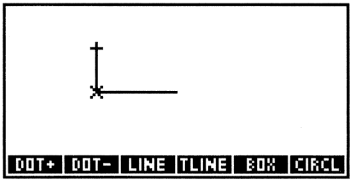
Example: Erase *PICT*, then use DOT+ to draw a horizontal line from the center halfway toward the left edge.



(↱) (PLOT) ERASE
(◀) (NXT) DOT+
(◀) (hold down)

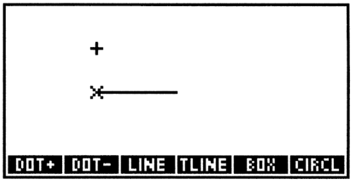
Turn off line-drawing. Then use TLINE to draw a vertical line from the current cursor position halfway to the top edge. (The first TLINE just sets the mark.)

19



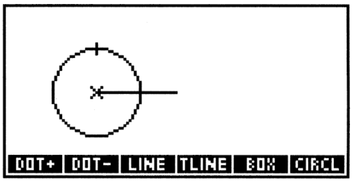
DOT+■
TLINE
(▲) (hold down)
TLINE

Toggle the line off.



TLINE

Draw a circle using the existing mark and the current cursor position.



CIRCL

Press (ATTN) to return to the stack display.

Adding Elements Using Commands

You can use commands to add graphical elements to *PICT*—either from the keyboard or in programs.

To add graphical elements using commands:

1. Enter the coordinates or other arguments required by the command.
2. Press the menu key for the command.

You can supply coordinate arguments in either user-unit form `(x,y)` or pixel form `{ #nx #ny }`.

19

Graphical Element Commands

Key	Programmable Command	Description
PRG DSPL (pages 1 and 2):		
LINE	LINE	Draws a line in *PICT* between the coordinates in levels 2 and 1.
TLINE	TLINE	Same as LINE except that pixels along the line are toggled on or off, rather than turned on.
BOX	BOX	Draws a box in *PICT* using two coordinate arguments as opposite corners.
ARC	ARC	Draws an arc in *PICT* centered at a coordinate (in level 4) with a given radius (in level 3) counterclockwise from θ_1 (in level 2) to θ_2 (in level 1). (The coordinate and radius must both use user-units or pixels.)
PIXON	PIXON	Turns on the pixel in *PICT* specified in level 1.
PIXOF	PIXOFF	Turns off the pixel in *PICT* specified in level 1.

Graphical Element Commands (continued)

Key	Programmable Command	Description
PIX?	PIX?	Returns 1 if the pixel specified in level 1 is on; returns 0 if the pixel is off.
PX→C	PX→C	Converts a pixel coordinate { $\#n_x$ $\#n_y$ } to a user-unit coordinate (x, y).
C→PX	C→PX	Converts a user-unit coordinate (x, y) to a pixel coordinate { $\#n_x$ $\#n_y$ }.

19

Working with Graphics Objects on the Stack

You can put graphics objects on the stack and store them in variables—just as you can with other types of objects. On the stack, a graphics object is displayed as

Graphic n × m

where n and m are the width and height in pixels.

When you put a graphics object from the stack into the command line, it's displayed as

GROB n m h

where GROB is the delimiter, n and m are the width and height in pixels, and h is the pixel pattern represented as hexadecimal digits (0–9 and A–F).

You can work with graphical elements using operations in the Graphics environment and using commands.

Using Stack Operations in the Graphics Environment

The following operations in the Graphics environment take a graphics object from the stack or return a graphics object to the stack. These operations aren't programmable.

Stack Operations in the Graphics Environment

Key	Description
REPL	Superimposes the graphics object from level 1 on *PICT*. The upper left corner of the graphics object is positioned at the cursor.
SUB	Puts on the stack the rectangular graphics object whose opposite corners are defined by the mark and cursor.
STO	Copies *PICT* to the stack as a graphics object.

19

Using Stack Commands for Graphics Objects

You can use commands to work with graphics objects and control the display—either from the keyboard or in programs.

To work with graphics objects on the stack:

1. Recall the graphics object or enter other arguments required by the command.
2. Press the menu key for the command.

You can supply coordinate arguments in either user-unit form (x, y) or pixel form { $\#n_x$ $\#n_y$ }.

19

Graphics Object Commands

Key	Programmable Command	Description
PRG DSPL:		
PVIEW	PVIEW	(*PICT* view.) Displays *PICT* with the specified coordinate (level 1) at the upper left corner of the graphics display. If the argument is an empty list, displays *PICT* centered in the display with scrolling mode activated.
SIZE	SIZE	For the graphics object in level 1, returns the width (level 2) and height (level 1) in pixels.
→GRO	→GROB	(To graphics object.) Converts an object (level 2) into a graphics object using real number n (0 to 3 from level 1) to specify the character size. The resultant graphics object is a string of small (n=1), medium (n=2), or large (n=3) characters. For n=0, the character size is the same as for n=3, except that for algebraic and unit objects, the resultant graphics object is the EquationWriter picture.
BLANK	BLANK	Creates a blank graphics object on the stack of size $\#n_x$ (in level 2) by $\#n_y$ (in level 1).
GOR	GOR	(Graphics-object OR.) Superimposes the level 1 graphics object onto the level 3 graphics object. The upper left corner of the level 1 graphics object is positioned at coordinates specified in level 2.
GXOR	GXOR	(Graphics-object XOR.) Same as GOR except that the level 1 graphics object appears normal on a light background and inverse on a dark background.

Graphics Object Commands (continued)

Key	Programmable Command	Description
REPL	REPL	(Replace.) Same as GOR except that the level 1 graphics object *overwrites* the level 3 graphics object where the level 1 graphics object is located.
SUB	SUB	(Subset.) Extracts a portion of a graphics object and returns it to the stack. It takes three arguments—a graphics object (level 3) and two coordinates (levels 2 and 1) that define the diagonal corners of the rectangle to be extracted.
→LCD	→LCD	(Stack to LCD.) Displays the graphics object from level 1 in the *stack* display, with its upper left pixel in the upper left corner of the display. It overwrites all of the display except the menu labels.
LCD→	LCD→	(LCD to stack.) Returns a graphics object to level 1 representing the current stack display.
FREEZ	FREEZE	"Freezes" one or more of three display areas so that they're not updated until a key press. (See "Using DISP FREEZE HALT...CONT for Input" on page 29-4.) Used with PVIEW in a program so that *PICT* persists in the stack display until a key press.
TEXT	TEXT	Displays the stack display.

19

Example: Program *PIE* on page 31-40 uses ARC and LINE to draw a pie chart. It then recalls *PICT* to the stack and executes GOR to merge a label with each slice of the pie chart.

Example: Program *WALK* on page 31-47 uses a custom graphical image in a program, executing GXOR in a loop structure to animate the image.

Using Stack Commands with PICT

You can use the name PICT as an argument to several graphics objects commands described above. For example, the SUB command accepts *PICT* as an argument, letting you define a region of *PICT* to return to the stack as a graphics object. This is the stack related equivalent of the SUB operation in the Graphics environment.

To put the name PICT on the stack:

- Press [PRG] DSPL PICT .

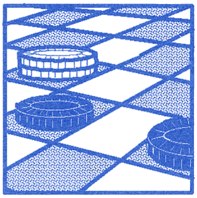
19 The PICT command puts the name PICT on the stack so you can access the *PICT* graphics object as if it were stored in a variable.

To work with PICT on the stack:

- To recall the *PICT* graphics object to the stack, press [PRG] DSPL PICT [↱][RCL].
- To store the graphics object in level 1 as the *PICT* graphics object, press [PRG] DSPL PICT [STO].
- To purge the contents of *PICT*, press [PRG] DSPL PICT [↰][PURGE].

20

Arrays



The HP 48 has extensive capabilities for entering and manipulating arrays. *Array* objects represent both vectors and matrices. A *vector* is a one-dimensional array. A *matrix* is a two-dimensional array.

20

This chapter covers these topics:

- Using the MatrixWriter application to enter and edit arrays.
- Using the command line to enter arrays.
- Doing arithmetic operations with arrays.
- Working with complex-number arrays.

Two-element and three-element vectors are particularly useful in engineering—they're covered in chapter 12, "Vectors."

Displaying Arrays

A matrix appears on the stack as numbers within nested `[ ]` delimiters. A pair of `[ ]` delimiters enclose the entire matrix—additional pairs enclose each row in the matrix. For example, here's a 3 × 3 matrix as it appears on the stack:

```
[[ 1 2 3 ]
 [ 3 4 5 ]
 [ 7 8 9 ]]
```

A vector (or column vector, mathematically equivalent to a one-column matrix) appears on the stack as numbers within one level of `[ ]` delimiters. For example, here's a 4-element vector as it appears on the stack:

```
[ 1 2 3 4 ]
```

The less-frequent row vector (a one-*row* matrix) appears on the stack as numbers within *two* pairs of `[ ]` delimiters. For example, `[[ 1 2 3 4 ]]` is how a 4-element row vector appears on the stack.

The current coordinate mode and angle mode affect how 2-dimensional and 3-dimensional vectors are displayed. See "Displaying 2D and 3D Vectors" on page 12-1.

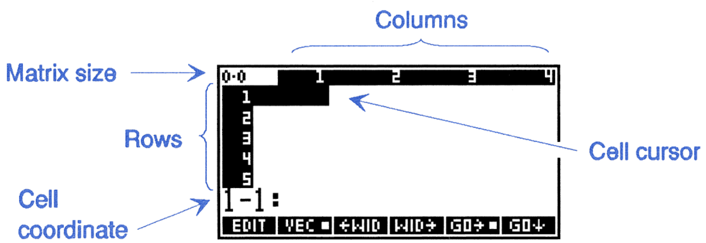
Entering Arrays

You can enter an array two ways, as described in this chapter:

- **MatrixWriter application.** A visual method of entering, viewing, and editing array elements.
- **Command line.** The basic object-entry method.

Using the MatrixWriter Application

The MatrixWriter application provides a special environment for entering, viewing, and editing arrays. The display shows array elements in individual cells arranged in rows and columns.



To enter a matrix using the MatrixWriter application:

1. Press (↱)(MATRIX) to display the MatrixWriter screen and menu.
2. For each number in the first row, enter the number and press (ENTER).
3. Press (▼) to mark the end of the first row.
4. For each number in the rest of the matrix, enter the number and press (ENTER).
5. After you've entered all of the numbers in the matrix, press (ENTER) to put the matrix on the stack.

While you're entering a number, the cell coordinate is replaced by the command line. When you press (ENTER) to store the value in the cell, the cell cursor normally advances to the next cell.

20

When you press (▼) at the end of the first row, it sets the number of columns in the matrix and moves the cursor to the beginning of the next row. You don't have to press (▼) again—the cell cursor automatically wraps to each new row.

If the displayed number is wider than the cell width, an ellipsis indicates "more to the right" (as in 1.2...). The default cell width is four characters.

Note the two uses of (ENTER): While you're using the command line for data entry, (ENTER) enters data into a cell. When a cell coordinate is displayed, (ENTER) enters the entire matrix onto the stack.

To enter a vector using the MatrixWriter application:

1. Press (↱)(MATRIX) to display the MatrixWriter screen and menu.
2. For each number in the vector, enter the number and press (ENTER).
3. After you've entered all of the numbers in the vector, press (ENTER) to put the vector on the stack.

For a vector, you normally use only the first row of data—so you don't need to press (▼).

To have more flexibility during data entry:

- To enter numbers into more than one cell at a time, press (SPC) to separate the numbers, then press (ENTER) to enter them all.
- To compute elements in the command line as you enter them, enter arguments and press command keys as required (press (SPC) to separate arguments), then press (ENTER) to compute the value and

put it in the cell. The commands aren't executed until you press (ENTER).

- To make the displayed cells narrower or wider, press ←WID or WID→.

Example: To enter 2.2^4 in a cell, press 2.2 (SPC) 4 (y^x) (ENTER).

Example: Enter the matrix

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

Select the MatrixWriter application.

(⮫) (MATRIX)

Key in the first element (cell 1-1).

2

Enter the value into the cell.

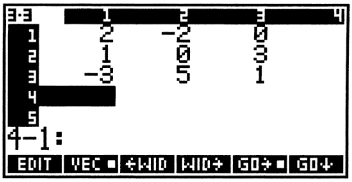
(ENTER)

Enter the rest of the first row.

2 (+/–) (SPC) 0 (ENTER)

Use (▼) to end the first row. Then, enter rest of the matrix.

(▼)
1 (SPC) 0 (SPC) 3 (SPC)
3 (+/−) (SPC) 5 (SPC) 1 (ENTER)



Enter the matrix onto the stack. (This matrix is used in a later example.)

(ENTER)

Using the Command Line

To enter a matrix using the command line:

1. Press (◀)([ ]) and (◀)([ ]) to type the delimiters for the matrix and for the first row.
2. Key in the first row. Press (SPC) to separate the elements.
3. Press (▶) to move the cursor past the] row delimiter.
4. Optional: Press (↱)(↵) (carriage return) to start a new row in the display.
5. Key in the rest of the matrix. You don't need [ ] delimiters for subsequent rows—they're added automatically later.
6. Press (ENTER).

To enter a vector using the command line:

1. Press (◀)([ ]) to type the delimiters for the vector.
2. Key in the vector elements. Press (SPC) to separate the elements.
3. Press (ENTER).

Example: Use the command lint to enter the matrix

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

Key in the delimiters and the first row.

(←)([ ]) (←)([ ]) 2 (SPC) 2 (SPC) 1

```
[[2 2 1◆]
ERASE DRAW AUTO XRNG YRNG INDEP
```

Move the cursor past the first] and key in the remaining values.

(▶) (→)(↵) 1 (SPC) 0 (SPC) 4
(→)(↵) 3 (SPC) 5 (SPC) 2

```
[[2 2 1]
1 0 4
3 5 2◆
ERASE DRAW AUTO XRNG YRNG INDEP
```

Enter the matrix onto the stack.

(ENTER)

```
1: [[ 2 2 1 ]
    [ 1 0 4 ]
    [ 3 5 2 ]]
ERASE DRAW AUTO XRNG YRNG INDEP
```


Viewing and Editing Arrays

To view an array using the MatrixWriter application:

1. View the array:
   - If the array is in level 1, press (▼).
   - If the array is stored in a variable, put the variable name in level 1 and press (→)(▼).
2. Press (ATTN) to return to the stack.

To edit an array you're viewing with the MatrixWriter application:

1. Press (◀) (▶) (▲) (▼) to move the cell cursor. (Use with (→) to move the cursor to the far end.)
2. Use the operations listed below to add or edit cells.
3. Press (ENTER) to save the changes (or press (ATTN) to discard them) and return to the stack.

To view or edit an array using the command line:

1. View the array:
   - If the array is in level 1, press (←)(EDIT).
   - If the array is stored in a variable, put the variable name in level 1 and press (→)(VISIT).
2. Optional: Make changes.

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

Operations in the MatrixWriter Environment

Key	Description
EDIT	Places contents of the current cell in the data entry line for editing. (Press ← EDIT to get the EDIT menu.) Press ENTER to save the changes, or press ATTN to discard them.
VEC	For one-row arrays, toggles between vector entry and matrix entry. If this key is "on" (VEC■), one-row arrays are entered into the command line as vectors (example: [1 2 3]). If it's "off" (VEC), one-row arrays are entered as matrices (example: [[1 2 3]]).
←WID	Narrows all cells so that one more column appears.
WID→	Widens all cells so that one fewer column appears.
GO→	Sets left-to-right entry mode. The cell cursor moves to the next *column* after data entry.
GO↓	Sets top-to-bottom entry mode. The cell cursor moves to the next *row* after data entry.
+ROW	Inserts a row of zeros at the current cursor position. (To insert a row at the bottom, see below.)
-ROW	Deletes the current row.
+COL	Inserts a column of zeros at the current cursor position. (To insert a column at the far right, see below.)
-COL	Deletes the current column.
→STK	Copies the current cell to level 1 of the stack.
↑STK	Activates the Interactive Stack.

If both GO→ and GO↓ are off (no ■ in either menu label), the cursor doesn't advance after an entry is made.

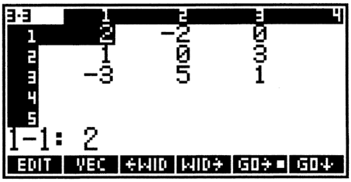
To add a column to the right of the last column, move the cursor to that column and enter a value. The rest of the column is filled with zeros. Use a similar procedure to add a row to the bottom.

Example: Change the matrix entered in the second previous example

$$\text{from} \quad \begin{bmatrix} 2 & -2 & 0 \\ 1 & 0 & 3 \\ -3 & 5 & 1 \end{bmatrix} \quad \text{to} \quad \begin{bmatrix} 2 & -2 & 4 & 0 \\ 1 & 0 & 1 & 3.1 \\ -3 & 5 & 3 & 1 \end{bmatrix}$$

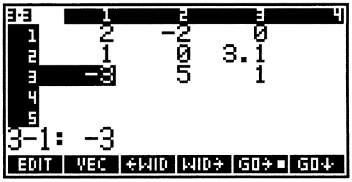
If the matrix is on the stack, bring it into level 1—otherwise, enter the matrix into level 1. Then view the matrix in the MatrixWriter environment. (This example assumes GO→■ is active.)

(⬅) (or enter the matrix)
(▼)



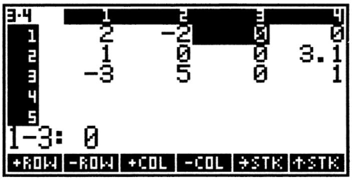
Edit element 2-3:

(▼) (▶) (▶)
EDIT (▶) .1 (ENTER)



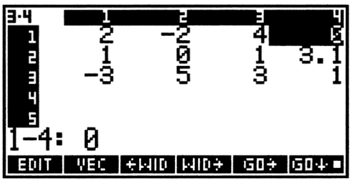
Insert a new column in front of column 3, and move the cell cursor to the top of the new column.

(▶) (▶) (NXT) +COL (▲) (▲)



Set top-to-bottom entry mode. Fill in the new column.

(NXT) GO↓
4 (SPC) 1 (SPC) 3 (ENTER)



Restore left-to-right entry mode, then enter the edited matrix.

GO→ (ENTER)

Calculating with Arrays

You can put arrays on the stack and perform mathematical operations on those arrays. The following tables summarize basic operations you can use. Other operations are listed under "More Matrix Commands" on page 20-16. 20

Arithmetic Operations for Vectors

Key	Programmable Command	Description
(+) (−)	+ −	**Addition and Subtraction.** Adds or subtracts two vectors that have the same number of elements. If either vector contains complex elements, the resulting vector is complex.
(×) (÷)	* /	**Multiplication and Division.** Multiplies or divides a vector by a real or complex number.
(MTH) VECTR:		
DOT	DOT	**Dot Product.** Returns the dot product of two vectors with the same number of elements.
CROSS	CROSS	**Cross Product.** Returns the cross product of two vectors with the same number of elements.
ABS	ABS	**Length.** Returns the length or magnitude of a vector. (Also in MTH PARTS menu.)

For examples of using DOT, CROSS, and ABS with vectors, see "Calculating with 2D and 3D Vectors" on page 12-8.

Arithmetic Operations for Matrices

Key	Programmable Command	Description
[1/x]	INV	**Inverse.** Calculates the inverse of a square matrix.
[+] [−]	+ −	**Addition and Subtraction.** Adds or subtracts two matrices that have the same dimensions.
[×] [÷]	* /	**Scalar Multiplication and Division.** Multiplies or divides each element in the array by a real or complex number. For division, the scalar must be in level 1.
[×]	*	**Matrix Multiplication.** Returns the product of the two arrays. The number of columns in the level 2 matrix must equal the number of rows in the level 1 matrix.

Example: Calculate the inverse of the matrix

$$\begin{bmatrix} 1 & 2 \\ 1 & 4 \end{bmatrix}$$

Enter the matrix—use the command line.

[⬅][[]] [⬅][[]] 1 [SPC] 2
[▶] 1 [SPC] 4 [ENTER]

```
1: [[ 1 2 ]
    [ 1 4 ]]
ERASE DRAW AUTO XRNG YRNG INDEP
```

Calculate the inverse.

[1/x]

```
1: [[ 2 -1 ]
    [ -.5 .5 ]]
ERASE DRAW AUTO XRNG YRNG INDEP
```

Example: Calculate the matrix product

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

Enter the first matrix.

(↱)(MATRIX)
2 (SPC) 2 (ENTER) (▼)
4 (SPC) 1 (SPC) 2 (SPC) 3 (ENTER)
(ENTER)

```
1: [[ 2 2 ]
    [ 4 1 ]
    [ 2 3 ]]
ERASE DRAW AUTO XRNG YRNG INDEP
```

Enter the second matrix.

(↱)(MATRIX)
2 (SPC) 2 (SPC) 1 (SPC) 4 (ENTER) (▼)
3 (SPC) 4 (SPC) 2 (SPC) 1 (ENTER)
(ENTER)

```
2: [[ 2 2 ] [ 4 1 ] …
1: [[ 2 2 1 4 ]
    [ 3 4 2 1 ]]
ERASE DRAW AUTO XRNG YRNG INDEP
```

20

Multiply the matrices.

(×)

```
1: [[ 10 12 6 10 ]
    [ 11 12 6 17 ]
    [ 13 16 8 11 ]]
ERASE DRAW AUTO XRNG YRNG INDEP
```

Arithmetic Operations for a Matrix and a Vector

Key	Programmable Command	Description
(×)	*	**Matrix-Vector Multiplication.** The number of columns in the matrix (level 2) must equal the number of elements in the vector (level 1). (The vector is treated as a column vector.)
(÷)	/	**Vector-Matrix Division.** The number of elements in the vector **y** (level 2) must equal the number of columns of the square matrix **X** (level 1). Returns the product $\mathbf{X}^{-1}\mathbf{y}$, often used to solve a system of linear equations.

Example: Calculate the product

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

Enter the matrix.

(↱)(MATRIX)
2 (SPC) 1 (SPC) 3 (ENTER) (▼)
4 (SPC) 2 (SPC) 2 (ENTER)
(ENTER)

```
1: [[ 2 1 3 ]
    [ 4 2 2 ]]
ERASE DRAW AUTO XRNG YRNG INDEP
```

Enter the 1×3 matrix as a vector, then do the multiplication.

(↰)([]) 3 (SPC) 1 (SPC) 1 (×)

```
1:              [ 10 16 ]
ERASE DRAW AUTO XRNG YRNG INDEP
```

20 **To solve a system of linear equations:**

1. Enter the n-element vector of constants.
2. Enter the $n \times n$ matrix of coefficients.
3. Press (÷) to get the n-element vector of variable values.

The system of linear equations $\mathbf{y} = \mathbf{Ax}$ must consist of n equations and n variables. The solution is calculated as $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$.

Example: Solve the following system of three linear, independent equations with three variables:

$$\begin{aligned} 3x + y + 2z &= 13 \\ x + y - 8z &= -1 \\ -x + 2y + 5z &= 13 \end{aligned}$$

Enter the constant vector.

(↱)(MATRIX) 13 (SPC) 1 (+/−) (SPC) 13
(ENTER) (ENTER)

```
1:          [ 13 -1 13 ]
ERASE DRAW AUTO XRNG YRNG INDEP
```

Enter the coefficient matrix.

(↱)(MATRIX)
3 (SPC) 1 (SPC) 2 (ENTER) (▼)
1 (SPC) 1 (SPC) 8 (+/−) (SPC)
1 (+/−) (SPC) 2 (SPC) 5 (ENTER)
(ENTER)

```
2:          [ 13 -1 13 ]
1: [[ 3 1 2 ]
    [ 1 1 -8 ]
    [ -1 2 5 ]]
ERASE DRAW AUTO XRNG YRNG INDEP
```

"Divide" the vector by the matrix.

(÷)

```
1:              [ 2 5 1 ]
ERASE DRAW AUTO XRNG YRNG INDEP
```

The values that satisfy the equations are $x = 2$, $y = 5$, and $z = 1$.

Calculating with Complex Arrays

Arrays can contain real numbers or complex numbers—but no other object types are allowed. A *complex array* is a vector or matrix that contains one or more complex-number elements.

You can use complex arrays for the arithmetic operations described in the previous section. If either argument is a complex array, the result is a complex array. For example, if you add a real matrix and a complex matrix, the result is a complex matrix.

20

You can use any command that manipulates real arrays to manipulate complex arrays—with the exception of the coordinate-mode-dependent commands (V→, →V2, and →V3). In addition, the commands listed below operate on complex arrays.

Commands for Manipulating Complex Arrays

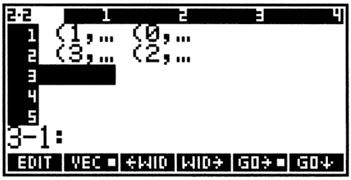
Keys	Programmable Command	Description
+/-	NEG	Returns an array in which each element is the negative of the argument array.
PRG OBJ (page 2):		
R→C	R→C	Combines two arrays into a complex array. The array in level 2 becomes the real part—the array in level 1 becomes the imaginary part.
C→R	C→R	Returns to levels 2 and 1 two arrays containing the real and imaginary parts of a complex array.
MTH PARTS:		
CONJ	CONJ	Returns the complex conjugate of a complex array—each element is conjugated.
RE	RE	Returns a real array consisting of the real parts of a complex array.
IM	IM	Returns a real array consisting of the imaginary parts of a complex array.

Example: Calculate the conjugate of the matrix

$$\begin{bmatrix} 1+3i & i \\ 3 & 2-4i \end{bmatrix}$$

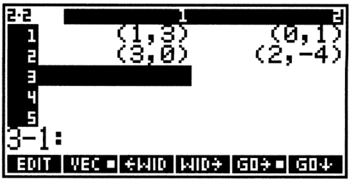
Select the MatrixWriter application and enter the complex numbers.

⮫ MATRIX
⮪ ( ) 1 SPC 3 ENTER
⮪ ( ) 0 SPC 1 ENTER ▼
3 ENTER
⮪ ( ) 2 SPC 4 +/- ENTER



Widen the columns to see the full entry.

WID→ WID→



Put the matrix onto the stack.

ENTER

Compute the conjugate.

MTH PARTS CONJ

Calculating with Algebraic Syntax

You can perform calculations with array elements using algebraic syntax. The array must be represented by a name in the symbolic expression or equation.

To enter an array element in a symbolic expression:

1. Inside the expression, enter the array name and press ↰ ( ).
2. Enter the subscripts for the element:
   - For a vector, enter one subscript.
   - For a matrix, enter two subscripts separated by ↰ ,.

Example: Enter a symbolic expression for the sum of all elements of a 4-element vector stored in variable *VECT*.

' ↱ Σ j ↰ = 1 ↰ , 4 ↰ , VECT ↰ ( ) j ENTER.

Example: Enter a symbolic expression for the sum of all elements of a 2 × 5 matrix stored in *MATR*.

[←] [EQUATION]
[→] [Σ] j [▶] 1 [▶] 2 [▶]
[→] [Σ] k [▶] 1 [▶] 5 [▶]
MATR [←] [( )] j [←] [,] k [▶]

Press [ENTER] to put the expression on the stack.

More Matrix Commands

Additional commands for creating and manipulating matrices and accessing individual elements (→ARRY, GET, GETI, OBJ→, PUT, and PUTI) are covered under "Manipulating Objects" on page 4-12.

The following commands perform other matrix operations. They're located in the MTH MATR menu ([MTH] MATR).

Command/Description	Example	
	Input	**Output**
ABS Frobenius (Euclidean) norm; square root of the sums of the squares of the absolute values of the elements.	`1: [[2 2]` `[2 2]]`	`1: 4`

Command/Description	Example	
	Input	Output

CNRM Column norm; maximum value (over all columns) of the sums of the absolute values of all elements in a column.	`1: [[1 2 3]`
	`[4 5 6]`
	`[7 8 9]]`
		`1: 18`

CON Constant; returns a constant real or complex array using the dimensions specified by a list $\{n\}$ or $\{n\ m\}$ or by an existing array.	`2: { 2 3 }`
	`1: 7`
	
	`2: [[1 2 3]`
	`[4 5 6]]`
	`1: 7`
		`1: [[7 7 7]`
		`[7 7 7]]`
		
		`1: [[7 7 7]`
		`[7 7 7]]`

DET Determinant; returns the determinant of a square matrix.	`1: [[1 2]` `[3 4]]`	`1: -2`

IDN Identity; returns an $n \times n$ (in level 1) identity matrix, or replaces the elements of the matrix in level 1.	`1: 2`
	
	`1: [[1 2]`
	`[3 4]]`
		`1: [[1 0]`
		`[0 1]]`
		
		`1: [[1 0]`
		`[0 1]]`

RDM Redimension; Redimensions an array. The new dimensions are in a list in level 1. Elements preserve the order of the source array.	`2: [[1 2 3]`
	`[4 5 6]`
	`[7 8 9]]`
	`1: { 3 2 }`
		`1: [[1 2]`
		`[3 4]`
		`[5 6]]`

RNRM Row norm; maximum value (over all rows) of the sums of the absolute values of all elements in a row.	`1: [[1 2 3]`
	`[4 5 6]`
	`[7 8 9]]`
		`1: 24`

TRN Transpose; transposition of the argument; an $n \times m$ matrix is replaced by an $m \times n$ matrix. (Complex entries are conjugated.)	`1: [[1 2 3]`
	`[4 5 6]]`
		`1: [[1 4]`
		`[2 5]`
		`[3 6]]`

CON, IDN, RDM, and TRN allow name arguments in place of the array argument. For example, evaluating the sequence `'A1' 7 CON` replaces the array stored in *A1* with a constant array of the same dimensions.

Advanced Topics Relating to Matrices

20

Improving the Accuracy of System Solutions

Because of unavoidable rounding errors during calculation, a numerically calculated solution **Z** is usually slightly in error. In most cases these errors will correspond to less than one count in the 12th digit of each element of **A** and **B**.

When additional accuracy is desired, the computed solution **Z** can usually be improved by *iterative refinement* (also known as *residual* corrections). Iterative refinement involves calculating a solution to a system of equations, then improving its accuracy using the residual associated with the solution.

To use iterative refinement:

1. Use ÷ to calculate a solution to the original system **AX** = **B**. (Call the solution **Z**, an approximation to **X**, in error by **E** = **X** − **Z**.)
2. Recall **B**, **A**, and **Z** to the stack (in that order), then use RSD (MTH MATR menu) to calculate the residual **R** as **B** − **AZ**.
3. Use ÷ to solve **AE** = **R** for **E**. (Call the solution **F**, an approximation to **E**.)
4. Use + to calculate **F** + **Z**, a new approximation to **X**.

For **F** + **Z** to be a better approximation to **X** than is **Z**, the residual **R** = **B** − **AZ** must be calculated to extended precision. The function RSD does this. You can repeat the refinement process, but most of the improvement occurs in the first refinement.

Example: This user program solves a matrix equation, including one refinement using RSD:

```
« → B A « B A / B A 3 PICK RSD A / + » »
```

This program takes two array arguments **B** and **A** from the stack, (the same as /) and returns the result array **Z**, which will be a refined approximation to the solution **X** over that provided by / itself.

Singular Matrices

A singular matrix is a square matrix that doesn't have an inverse. You normally get an error if you use [1/x] to find the inverse of a singular matrix—or use [÷] to solve a system of linear equations having a singular coefficient matrix.

Because of unavoidable rounding errors, a calculated matrix may be singular, even though the *theoretical* result without rounding might *not* be singular. If you set flag −22 (Infinite Result Exception), you won't get an error if you use [1/x] or [÷] with a singular matrix. Instead, the HP 48 perturbs the singular matrix by an amount that's usually small compared to rounding error. The calculated result corresponds to that for a nonsingular matrix close to the original, singular matrix.

20

Over-Determined and Under-Determined Systems

An under-determined system of linear equations contains more variables than equations, and the coefficient array has fewer rows than columns. The following program solves an under-determined system $\mathbf{AX} = \mathbf{B}$ using the Moore-Penrose technique: $\mathbf{X} = \mathbf{A}^T(\mathbf{AA}^T)^{-1}\mathbf{B}$. The program requires as input the vector **B** in level 2 and the matrix **A** in level 1.

```
« → B A
  « A TRN
    B A A TRN * / *
  »
»
```

An over-determined system contains fewer variables than equations. The next program solves an over-determined system using the least squares method: $\mathbf{X} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{B}$. Like the previous program, its input is **B** in level 2 and **A** in level 1.

```
« → B A
  « A TRN B *
    A TRN A * /
  »
»
```

20

21

Statistics

The Statistics application enables you to calculate single-sample and paired-sample statistics. It also enables you to draw scatter plots, bar charts, and frequency histograms.

21

This chapter shows you how to calculate:

- Total, mean, maximum, and minimum.
- Sample standard deviation and covariance.
- Correlation coefficient.
- Curve-fitting with four models (linear, logarithmic, exponential, power).
- Summary statistics.
- Upper-tail probabilities for various test statistics.

Press (←)(STAT) to display the first page of the STAT menu. If there is any *current statistical data*, a message in the display shows the last values entered.

Organizing Statistical Data

Statistical data for the HP 48 is organized in the form of a matrix. The matrix contains a row for each *data point* and a column for each *variable* measured at that point.

	$\mathbf{var}_1$	$\mathbf{var}_2$	...	$\mathbf{var}_m$
$\mathbf{point}_1$	x_{11}	x_{12}	...	x_{1m}
$\mathbf{point}_2$	x_{21}	x_{22}	...	x_{2m}
⋮	⋮	⋮		⋮
$\mathbf{point}_n$	x_{n1}	x_{n2}	...	x_{nm}

21

Setting Up the Current Statistical Matrix

The Statistics application uses the data stored in the *current statistical matrix*. It's stored in reserved variable *ΣDAT*. You change the current statistical matrix each time you work with a different set of data. (Because *ΣDAT* is a variable, you can have a different current statistical matrix for each directory in memory.)

Entering Statistical Data

You can enter statistical data one point at a time, or you can create a complete matrix of data and make it the current statistical matrix.

To clear the current statistical matrix:

- Press ⬅ STAT CLΣ .

To key in statistical data with only one variable:

1. Press ⬅ STAT.
2. Press CLΣ to clear previous data.
3. For each point, enter the value and press Σ+ .

To key in statistical data with several variables:

1. Press ⬅ STAT.
2. Press CLΣ to clear previous data.
3. For the *first point*, create a vector (with [ ] delimiters) containing all the variable values for the point. (Press SPC to separate the values in the vector.)
4. Press Σ+ to enter the point.

5. For each remaining point, enter the variable values and press Σ+ . You can enter the values for each point as individual numbers *or* as a vector.

The first point defines the number of variables. All other data must have the same number of variables.

To create a matrix and make it the current statistical matrix:

1. Create the matrix and put it in level 1. You can use the MatrixWriter application, for example.
2. Press (◄)(STAT).
3. Store the matrix:
   - To store a copy of the matrix for future use, press NEW , type a name for it without pressing (α), and press (ENTER).
   - To not store a copy, press STOΣ .

21

The matrix itself is stored in the named variable, and the variable name is stored in *ΣDAT*. (If you don't enter a name, the matrix itself is stored in *ΣDAT*.)

Example: The following table lists the consumer price index (CPI), producer price index (PPI), and unemployment rate (UR) for the United States over a 5-year period. Enter the data.

Year	CPI	PPI	UR
1	9.1	9.2	8.5
2	5.8	4.6	7.7
3	6.5	6.1	7.0
4	7.6	7.8	6.0
5	11.5	19.3	5.8

Set 2 FIX display mode, start the Statistics application, and clear any previous statistical data.

(◄)(MODES) 2 FIX
(◄)(STAT) CLΣ

```
No current data. Enter
data point, press Σ+
```

Key in the data for year 1. Remember—enter the first data point as a vector.

(◄)([]) 9.1 (SPC) 9.2 (SPC) 8.5

```
[9.1 9.2 8.5◆
Σ+ | CLΣ | NEW |EDITΣ|STOΣ| CAT
```

Enter the data into the statistical matrix.

Σ+

```
ΣDAT(1)=[ 9.10 9.20 8…
ΣDAT(2)=
```

Enter the rest of the data. After the first row, you can enter a row as simple numbers.

5.8 (SPC) 4.6 (SPC) 7.7 Σ+
6.5 (SPC) 6.1 (SPC) 7 Σ+
7.6 (SPC) 7.8 (SPC) 6 Σ+
11.5 (SPC) 19.3 (SPC) 5.8 Σ+

```
ΣDAT(5)=[ 11.50 19.30…
ΣDAT(6)=
```

21

Editing Statistical Data

To revise the last data point:

1. Press (↰) Σ+ (the Σ− command) in the STAT menu to delete the last data point.
2. Optional: To revise the deleted data point and reenter it, press (↰)(EDIT), make the changes, and press (ENTER)—then press Σ+.

The Σ− command removes the last data point in *ΣDAT*, such as from the most recent Σ+ operation. The deleted data point is returned to level 1.

To edit any data point:

1. Press EDITΣ in the STAT menu to activate the MatrixWriter environment.
2. Edit any of the data points.
3. Press (ENTER) to save the changes (or press (ATTN) to discard them).

Summary of Data-Entry STAT Menu Operations

The first page of the STAT menu contains keys for entering and manipulating data. The other pages, described throughout this chapter, contain commands for doing calculations and drawing graphs.

The STAT Menu—Data-Entry Operations

Key	Programmable Command	Description
⬑ STAT:		
Σ+	Σ+	Enters data from the stack into the current statistical matrix.
⬑ Σ+	Σ−	Deletes the last data point from the statistical matrix and returns it to the stack.
CLΣ	CLΣ	Clears the current statistical matrix.
NEW		Takes a matrix from level 1, prompts for a variable name, stores the matrix in that variable, and makes that matrix the current statistical matrix.
EDITΣ		Places the current statistical matrix in the MatrixWriter environment for editing. Press ENTER to save the changes, or press ATTN to discard them.
STOΣ	STOΣ	Stores the matrix or name in level 1 as the current statistical matrix.
⮡ STOΣ	RCLΣ	Recalls the current statistical matrix to level 1.
CAT		Displays the catalog of matrices and subdirectories in the current directory.
⬑ REVIEW		Redisplays the status message relating to the last data entered.

21

Using the Statistics Catalog

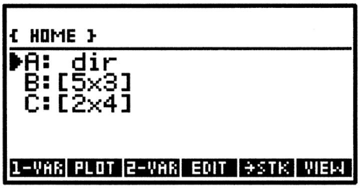
The Statistics Catalog enables you to specify any existing matrix as the current matrix. It's a special environment in which the keyboard is redefined and limited to specialized operations.

To select a matrix from the Statistics Catalog:

1. Press ⬅ STAT CAT .
2. Press ▲ and ▼ to move the pointer to the desired entry in the list.
3. To make the matrix the current matrix, press 1-VAR, 2-VAR, or PLOT —see the table below.

The Statistics Catalog lists all the variables in the current directory that contain matrices and all subdirectories in the current directory.

Catalog pointer →



To exit the Statistics Catalog without selecting a matrix:

- Press ATTN.

You can use the following operations to manipulate the entry you select.

Operations in the Statistics Catalog

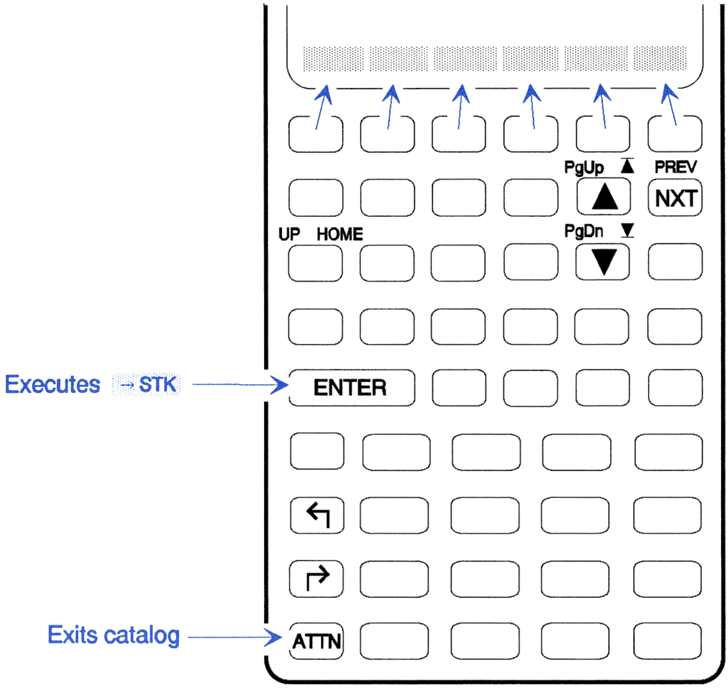
Key	Description
1-VAR	Makes the selected entry the current statistical matrix, leaves the catalog, and displays page 2 of the STAT menu (for calculating single-sample statistics).
PLOT	Makes the selected entry the current statistical matrix, leaves the catalog, and displays page 3 of the STAT menu (for plotting data).
2-VAR	Makes the selected entry the current statistical matrix, leaves the catalog, and displays page 4 of the STAT menu (for calculating paired-sample statistics).

Operations in the Statistics Catalog (continued)

Key	Description
EDIT	Places the selected entry in the MatrixWriter environment for editing. Press [ENTER] to save the changes, or press [ATTN] to discard them.
→STK	Copies the matrix to the stack.
VIEW	Lets you view the contents of the entry. If the entry is a subdirectory, switches to that subdirectory.
ORDER	Moves the selected matrix to top of the catalog.
PURG	Purges the entry (and its corresponding variable).
[NXT]	Selects the next page of Statistics Catalog operations.
[←][PREV]	Selects the previous page of Statistics Catalog operations.
[▲]	Moves the catalog pointer up one level. When prefixed with [←], moves the catalog pointer up one page ([←][PgUp] in the following illustration). When prefixed with [→], moves the catalog pointer to the top of the catalog ([→][⯭] in the illustration).
[▼]	Moves the catalog pointer down one level. When prefixed with [←], moves the catalog pointer down one page ([←][PgDn] in the following illustration). When prefixed with [→], moves the catalog pointer to the bottom of the catalog ([→][⯯] in the illustration).
[ENTER]	Executes →STK (copies matrix to stack). If the entry is a subdirectory, switches to that subdirectory, giving access to any matrices there.
[←][UP]	Switches to the parent directory.
[→][HOME]	Switches to the *HOME* directory.
[ATTN]	Exits the catalog.

21

The redefined keyboard looks like this:



Calculating Single-Variable Statistics

If your statistical data measures a *sample of a population*, you calculate sample statistics. If, however, your data measures the *entire population*, you calculate population statistics.

Getting Sample Statistics

Use commands in page 2 of the STAT menu to calculate single-sample statistics. Each command returns a vector containing m numbers, where m is the number of columns in the matrix. (If m=1, where each data point consists of only one number, the commands return one number.) For example, if you have a 3-column matrix in ΣDAT, MEAN returns a 3-element vector containing the mean of each column.

The STAT Menu—Single-Sample Statistics Commands

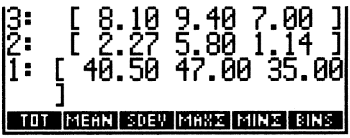
Key	Programmable Command	Description
(←)(STAT) (page 2):		
TOT	TOT	Total.
MEAN	MEAN	Mean (average).
SDEV	SDEV	Sample standard deviation.
MAXΣ	MAXΣ	Maximum value.
MINΣ	MINΣ	Minimum value.
BINS	BINS	Calculates frequencies using the independent-variable column (XCOL) of *ΣDAT*. Takes as its arguments the minimum x-value (level 3), the width of each bin in user units (level 2), and the number of bins, n (level 1). Returns a "bins" matrix and an "excess" vector—see below.
	VAR	Variance. (Command must be typed.)

The output of BINS is an $n \times 1$ "bins" matrix and a 2-element "excess" vector:

Level 2: `[[` n_1 `]` `[` n_2 `]` ... `[` n_n `]]`
Level 1: `[` n_{low} n_{high} `]`

Example: For the CPI data entered in the previous example, calculate the means, standard deviations, and totals of the CPI, PPI, and UR data.

(NXT) (if necessary)
MEAN
SDEV
TOT



Getting Population Statistics

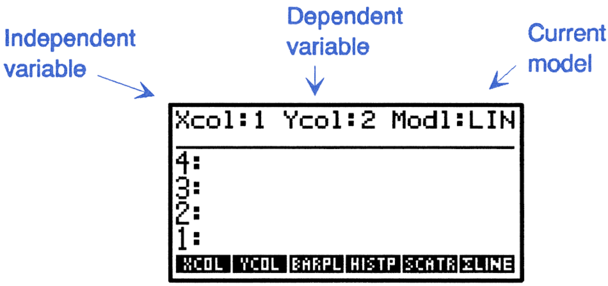
If you calculate the standard deviation or covariance for your data using SDEV or COV, it's computed assuming the data measures a *sample* of the population. If, however, the data measures the *entire* population, you can use the result to calculate the population statistics.

To calculate population standard deviation or covariance:

1. Press MEAN to calculate the mean of the data.
2. Press Σ+ to add the mean data point to *ΣDAT*.
3. Press SDEV or COV to calculate the population statistics.
4. Press ← Σ+ to remove the mean data point from *ΣDAT* (the Σ− command).

Calculating Paired-Sample Statistics

Use commands in pages 3 and 4 of the STAT to compute paired-sample statistics. When these pages of the STAT menu are displayed, the status message at the top of the display indicates the column designations for the independent (x) and dependent (y) variables and the current statistical model.



To calculate paired-sample statistics:

1. Enter the column number for the independent variable and press XCOL.
2. Enter the column number for the dependent variable and press YCOL.
3. Press MODL, then press the menu key for the desired statistical model.
4. Use commands on page 4 of the STAT menu to calculate paired-sample statistics—see the listing below. To calculate predicted values, press LR first, then use PREDX or PREDY.

You can choose one of four statistical models: LIN (LINFIT: linear), LOG (LOGFIT: logarithmic), EXP (EXPFIT: exponential), or PWR (PWRFIT: power). If you press BEST (BESTFIT), the HP 48 selects the model for which the correlation has the largest absolute value—or, if any data is negative or zero, LINFIT is selected.

You can also press SCATR to draw a scatter plot of the data—see the next section.

21

The STAT Menu—Paired-Sample Statistics Commands

Key	Programmable Command	Description
⬅ STAT (pages 3 and 4):		
XCOL	XCOL	Takes a column number as its argument, and makes that column the independent variable. (➡ XCOL returns the XCOL column number to level 1.)
YCOL	YCOL	Takes a column number as its argument, and makes that column the dependent variable. (➡ YCOL returns the YCOL column number to level 1.)
ΣLINE	ΣLINE	Returns the expression representing the best fit line according to the current model.

The STAT Menu—Paired-Sample Statistics Commands (continued)

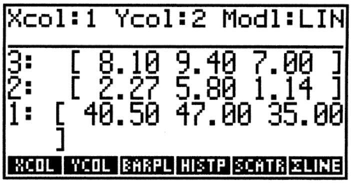
Key	Programmable Command	Description
LR	LR	Using the current model, computes the linear regression for the selected independent and dependent variables, and returns the intercept (level 2) and slope (level 1). Also, stores the intercept and slope values in ΣPAR.
PREDX	PREDX	Takes as its argument a value for the dependent variable, and computes a predicted value for the independent variable. (You must execute LR some time before PREDX.)
PREDY	PREDY	Takes as its argument a value for the independent variable, and computes a predicted value for the dependent variable. (You must execute LR some time before PREDY.)
CORR	CORR	Correlation (computed according to the current model).
COV	COV	Sample covariance (computed according to the current model).
MODL		Displays the menu for selecting a model: linear, exponential, power, or logarithmic. Selection is stored in ΣPAR.

When you execute these operations, the status message (x, y, and model) is erased. You can press ⬅ (REVIEW) to redisplay it.

Example: Using the CPI data from the previous examples in this chapter, calculate the correlation and covariance between CPI and PPI.

Make sure columns 1 and 2 are the x- and y-variables.

(NXT) (if necessary)
1 XCOL 2 YCOL (if necessary)



Calculate the correlation coefficient and covariance.

(NXT) CORR
COV

Example: Using the CPI data from the previous examples in this chapter, predict the PPI value for a CPI value of 8.5. (This example assumes you've already set the x column to 1, the y column to 2.)

Make sure the statistical model is linear. Then calculate the linear regression statistics.

(NXT) (if necessary)
MODL LIN (if necessary)
LR

Now calculate the predicted PPI (y) value.

8.5 PREDY

Plotting Statistical Data

You can plot statistical data three ways:

- **Scatter Plot.** For two variables, their values at each data point are depicted by a dot in the x-y plane.
- **Bar Chart.** For one variable, its value at each sequential data point is shown by a vertical bar.
- **Histogram.** For one variable, the number of times its value falls within certain ranges—called *bins*—is depicted by a vertical bar.

The Statistics application provides commands for plotting statistical data with relative ease. For more plotting control, the Plot application lets you specify additional parameters for statistical plots—see "Drawing Statistical Plots" on page 19-21.

Plotting Scatter Plots

A scatter plot shows the relationship between two variables by plotting a point at each x-y coordinate pair. For variables that are statistically correlated, the points should cluster along a curve representing the statistical model.

To draw a scatter plot:

1. Enter the column number for the x-axis variable and press XCOL in the STAT menu.
2. Enter the column number for the y-axis variable and press YCOL in the STAT menu.
3. Press SCATR in the STAT menu.
4. Optional: Press FCN to draw the curve for the current statistical model.
5. Press (ATTN) to return to the Statistics application.

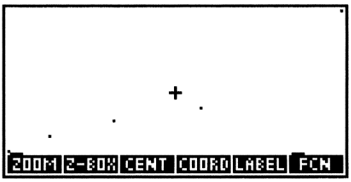
To change the statistical model:

1. Press MODL in the STAT menu.
2. Press the menu key for the desired model.

Example: Using the CPI data from the previous examples in this chapter, draw a scatter plot of PPI versus CPI, then plot the statistical model. (This example assumes you've already set the x column to 1, the y column to 2, and the model to linear.)

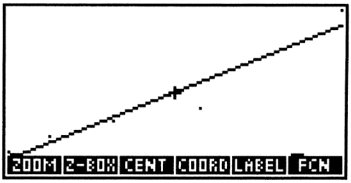
Plot a scatter plot of the data.

(↰)(PREV) (if necessary)
SCATR



Draw the best straight line for the data.

FCN



Press [ATTN] [↰] [MODES] STD to change the display mode back to Standard.

Plotting Bar Charts

A bar chart shows the values of one variable in the order they appear in the statistical matrix.

To draw a bar chart:

1. Enter the column number you want to plot and press XCOL in the STAT menu.
2. Press BARPL in the STAT menu.
3. Press [ATTN] to return to the Statistics application.

BARPLOT plots a bar chart of the specified column in ΣDAT. If you don't specify a column, the first column in ΣDAT is used. Data can be positive or negative, resulting in bars above or below the x-axis.

Example: Records from a gasoline station show the following relationship between the monthly percentage changes in gasoline price and amount sold over a 4-month period:

Month	Price (% Change)	Sales (% Change)
1	+3.5	−1.2
2	+9.3	−2.6
3	−6.5	+6.1
4	+2.0	−0.4

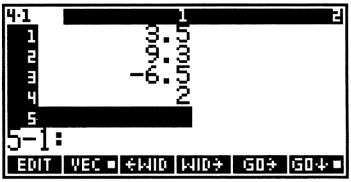
Enter the price and sales data using the MatrixWriter application, and then plot bar charts for the percentage change in price and the percentage change in sales.

Start the MatrixWriter application.

[→] [MATRIX]

Enter the price data (vertical entry order).

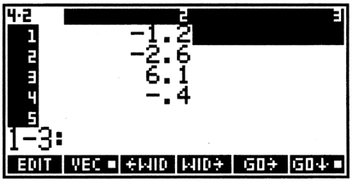
GO↓ 3.5 [SPC] 9.3 [SPC]
6.5 [+/-] [SPC] 2 [ENTER]



21

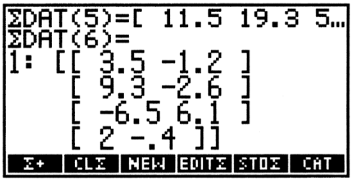
Enter the sales data.

[▶] 1.2 [+/-] [SPC] 2.6 [+/-] [SPC]
6.1 [SPC] .4 [+/-] [ENTER]



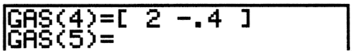
Enter the matrix onto the stack and start the Statistics application.

GO→ [ENTER]
[←] [STAT]



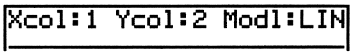
Name the matrix and make it the current matrix.

NEW GAS [ENTER]



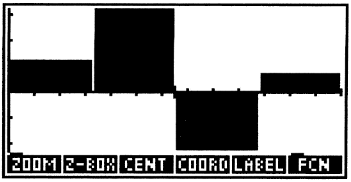
Select the column for percentage change in price (the first column in the statistical matrix).

[NXT] [NXT] 1 XCOL



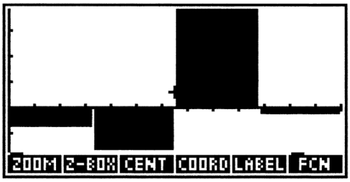
Draw the bar chart for percentage change in price.

BARPL



Select the column for percentage change in sales (the second column in the matrix) and draw a bar chart for it.

[ATTN] 2 XCOL BARPL



Press [ATTN] to return to the Statistics application.

Plotting Histograms

A histogram divides the range of values of one variable into a number of *bins* and for each bin shows the number of data points for which the variable value falls within the bin. HISTPLOT shows *relative frequency*—the maximum y value is the total number of data points.

To draw a histogram:

1. Enter the column number you want to plot and press XCOL in the STAT menu.
2. Press HISTP in the STAT menu.
3. Press [ATTN] to return to the Statistics application.

HISTPLOT automatically uses 13 bins. To change the number of bins, enter the number and press RES in the PLOTR menu ([↱][PLOT]). To use the default number of bins, press 0 RES.

To draw a histogram with specified bins:

1. Optional: If the statistical data isn't named, press [↱] STOΣ NEW, enter a name, and press [ENTER].
2. Enter the column number you want to plot and press XCOL in the STAT menu.

3. Enter the minimum x-value to use (the lower bound of the range) and press ENTER.
4. Enter the width of each bin (positive real number) and press ENTER.
5. Enter the number of bins you want and press ENTER.
6. Press BINS in the STAT menu.
7. Press ◆ (or ↰ DROP) to drop the out-of-range data.
8. Press STOΣ to store the frequency data as the current statistical data.
9. Press BARPL to plot the frequency data.
10. Press ATTN to return to the Statistics application.

21 Notice that the original statistical matrix is replaced in these steps. You can review the frequency data before you store and plot it. If it's not what you want, you can start the steps over. The y-axis of the "histogram" plotted by BARPL is scaled to the maximum frequency—not to the number of original data points.

Summary of Plotting Commands

Use commands in page 3 of the STAT menu to plot single- and paired-sample statistics. When this page of the STAT menu is displayed, the status message at the top of the display indicates the column designations for the independent (x) and dependent (y) variables and the current model.

The STAT Menu—Plotting Commands

Key	Programmable Command	Description
(⬅)(STAT) (page 3):		
XCOL	XCOL	Takes a column number as its argument, and designates that column as the independent variable.
YCOL	YCOL	Takes a column number as its argument, and designates that column as the dependent variable.
BARPL	BARPLOT	Draws a bar chart using the x-column. Autoscaled.
HISTP	HISTPLOT	Draws a frequency histogram using the x-column. Autoscaled.
SCATR	SCATRPLOT	Plots the (x,y) points using the designated x- and y-columns, and optionally draws the best line using the current model. Autoscaled.

When you execute some of these operations, the status message (x, y, and model) is erased. You can press (⬅)(REVIEW) to review the status information—hold down (REVIEW) to prolong the status display.

21

Calculating Summation Statistics

Use the commands in page 5 of the STAT menu to calculate summation statistics. Use XCOL and YCOL (in page 3 of the STAT menu) to designate x and y.

Summation Statistics Commands

Key	Programmable Command	Description
(←)(STAT) (page 5):		
ΣX	ΣX	Returns the sum of the entries in the x (independent) column of ΣDAT.
ΣY	ΣY	Returns the sum of the entries in the y (dependent) column of ΣDAT.
ΣX^2	ΣX^2	Returns the sum of the squares of the x-column entries of ΣDAT.
ΣY^2	ΣY^2	Returns the sum of the squares of the y-column entries of ΣDAT.
ΣX*Y	ΣX*Y	Returns the sum of the products of corresponding x and y columns in ΣDAT.
NΣ	NΣ	Returns the number of rows in ΣDAT.

Calculating Test Statistics

Use the commands in the PROB (probability) menu ((MTH) PROB) to calculate combinations, permutations, factorials, random numbers, and upper-tail probabilities of various test statistics.

Test statistics are calculated using values you enter on the stack—they do *not* use the statistical data stored in ΣDAT.

Only upper-tail probabilities are covered here—for the other topics, see "Factorial, Probability, and Random Numbers" on page 9-13.

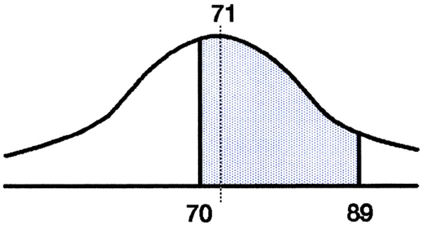
Test Statistics Commands

Keys	Programmable Command	Description
(MTH) PROB (page 2):		
UTPC	UTPC	**Upper-tail chi-square distribution.** Takes the degrees of freedom from level 2 and a real number (x) from level 1, and returns the probability that a χ^2 random variable is greater than x.
UTPF	UTPF	**Upper-tail f distribution.** Takes the numerator degrees of freedom from level 3, the denominator degrees of freedom from level 2, and a real number (x) from level 1, and returns the probability that a Snedecor's F random variable is greater than x.
UTPN	UTPN	**Upper-tail normal distribution.** Takes the mean from level 3, the variance from level 2, and a real number (x) from level 1, and returns the probability that a normal random variable is greater than x for a normal distribution.
UTPT	UTPT	**Upper-tail t distribution.** Takes the degrees of freedom from level 2 and a real number (x) from level 1, and returns the probability that the Student's t random variable is greater than x.

21

Note that, when used as an argument for these commands, the number of degrees of freedom must be between 0 and 499. Also, in the calculations, the degrees of freedom are rounded to the nearest integer.

Example: The scores on a final exam approximate a normal curve with a mean of 71 and standard deviation of 11. What percentage of the students scored between 70 and 89?



First, calculate the probability that a student chosen at random obtained a score greater than 70. (Square the standard deviation to get the variance.)

(MTH) PROB (NXT)
71 (ENTER)
11 (⬅)(x^2)
70 UTPN

Now, do the same calculation for a score of 89.

(➡)(LAST ARG) (◆)
89 UTPN

Subtract the two values. About 49% of the students scored between 70 and 89.

(−)

Understanding the Statistics Parameter Variable

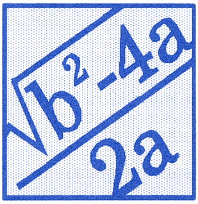
The HP 48 uses a built-in statistics parameter variable named *ΣPAR* to store the statistics parameters. You normally control the parameters using the XCOL, YCOL, LR, and MODL commands in the STAT menu. Because *ΣPAR* is a variable, you can have a different *ΣPAR* in every directory. *ΣPAR* contains a list with the following objects:

{ *independent-col dependent-col intercept slope model* }

The default contents are `{ 1 2 0 0 LINFIT }`. 21

22

Algebra



This chapter shows you how to manipulate a symbolic expression or equation much as you would on a piece of paper. You can solve symbolically for a variable, and you can simplify and rearrange an expression or equation.

In this chapter, the term *algebraic* is used to mean an algebraic object—a symbolic expression or equation. The terms *expression* and *equation* are used only where the distinction between these two forms of an algebraic object is important.

Finding Symbolic Solutions

A common goal of algebraic manipulation of an expression or equation is to "solve for" a variable symbolically—that is, to express one variable in terms of the other variables and numbers in the expression or equation. You can solve symbolically using these commands:

- **ISOL.** Solves for a variable that appears only once in any type of expression or equation.
- **QUAD.** Solves for a variable that appears in a quadratic expression or equation.

Comparison of Commands for Symbolic Solutions

ISOL Command	QUAD Command
Variable appears only once. Variable can be any order. Variable can be the argument of a nonlinear function (such as SIN).	Variable can appear several times—no rearranging required. Variable must not be higher than second order for an exact solution.

Isolating a Single Variable

22

To solve for a variable that appears only once:

1. Enter the algebraic on the stack.
2. Enter the name of the variable (with ' delimiters).
3. Press [⬅] [ALGEBRA] ISOL .

The ISOL command isolates a single occurrence of a variable in an algebraic—it returns an equation that represents a symbolic solution of the algebraic:

' *variable*=*expression* '

If the algebraic is an expression (it has no =), the expression is treated as an equation of the form ' *expression*=0 '.

The variable to be isolated can be the argument of a function *only if the HP 48 has an inverse for that function*. Functions for which the HP 48 has inverses are called *analytic* functions in this manual. For example, you can isolate X in an algebraic containing TAN(X) or LN(X) because TAN and LN have inverses (ATAN and EXP). However, you cannot isolate X in an algebraic containing IP(X). The operations index in appendix G identifies the HP 48 analytic functions.

If there is more than one solution for the algebraic, ISOL includes an "s" (sign) variable to give a general solution. See "Getting General and Principal Solutions" on page 22-5.

Example: Use ISOL to isolate A in the equation:

$$T = \sqrt{\frac{X+B}{X+A}}$$

Key in the equation.

[⬅] [EQUATION] T [⬅] [=]
[√x] [▲] X [+] B [▶]
X [+] A

```
T=√((X+B)/(X+A))
UTPC UTPF UTPN UTPT
```

Enter the equation. Then enter the name of the variable and isolate it.

[ENTER]
['] A [ENTER]
[⬅] [ALGEBRA] ISOL

```
1:    'A=(X+B)/SQ(T)-X'
COLCT EXPA ISOL QUAD SHOW TAYLR
```


Example: Enter the expression `'A*(B+2*X)-C'` and variable `'X'`, then press ISOL to isolate X. You get `'X=(C/A-B)/2'`.

Solving Quadratic Equations

To solve for a variable in a quadratic:

1. Enter the quadratic equation or expression on the stack.
2. Enter the name of the variable (with ' delimiters).
3. Press [⬅] [ALGEBRA] QUAD.

The QUAD command solves any algebraic that is up to second order in the unknown variable. The command is named for its ability to solve second-order (quadratic) algebraics, but you can also use QUAD to solve first-order (linear) algebraics. It returns an equation of the form

'*variable*=*expression*'

If you supply an equation that is *not* first or second order in the variable to be solved for, QUAD transforms the equation into a second order polynomial *approximation* and then solves that quadratic.

If the algebraic contains other variables, they must not exist in the current directory if you want those variables to be included in the solution as formal (symbolic) variables. If they exist in the current directory, QUAD evaluates them.

If the algebraic is an expression, the expression is treated as an equation of the form '*expression*=0'.

If there is more than one solution for the algebraic, QUAD includes "s" (sign) and "n" (integer) variables to give a general solution. See "Getting General and Principal Solutions" on page 22-5.

Example: Quadratic. Solve for x in the expression $x^2 - x - 6$.

(This example assumes that variable X does not exist in the current directory—you can press ['] X [←][PURGE].)

Enter the expression and the name of the variable.

['] X [y^x] 2 [−] X [−] 6 [ENTER]
['] X [ENTER]

```
2:              'X^2-X-6'
1:                    'X'
COLCT EXPA ISOL QUAD SHOW TAYLR
```


Solve for the variable.

[←][ALGEBRA] QUAD

```
1:         'X=(1+s1*5)/2'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

The solution contains the variable *s1*, which represents an arbitrary + or − sign. Copy the expression. Then evaluate it for *s1* = 1. (To key in *s1*, press [α] [←] S1.)

[ENTER]
1 ['] s1 [STO]
[EVAL]

```
1:                  'X=3'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Now evaluate the expression for *s1* = −1

1 [+/−] ['] s1 [STO]
[←][SWAP]
[EVAL]

```
2:                  'X=3'
1:                 'X=-2'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Example: Quadratic with Other Variables. Solve for x in the equation $2x^2 - 4x + c = 0$.

(This example assumes that variables X and C do not exist in the current directory—you can press [←][{ }] X [SPC] C [ENTER] [←][PURGE].)

Enter the equation and the variable name X.

['] 2 [×] X [y^x] 2
[−] 4 [×] X [+] C [←][=] 0
[ENTER] ['] X [ENTER]

```
2:        '2*X^2-4*X+C=0'
1:                    'X'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Solve for the variable in terms of C.

[←] [ALGEBRA] QUAD

```
1: 'X=(4+s1*√(16-8*C))
   /4'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Copy the expression. Calculate the roots for $c = 3$. The roots are $1 \pm 0.7071i$.

[ENTER]
3 ['] C [STO]
1 ['] s1 [STO] [EVAL]
1 [+/-] ['] s1 [STO]
[←] [SWAP] [EVAL]

```
2: 'X=(1,.70710678118…
1: 'X=
   (1,-.707106781188)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

22

Example: Linear Equation. Use QUAD to solve for x (which appears twice) in the equation $3(x + 2) = 5(x - 6)$.

Key in the equation.

['] 3 [×] [←] [()] X [+] 2 [▶]
[←] [=] 5 [×] [←] [()] X [-] 6
[ENTER]

```
1:    '3*(X+2)=5*(X-6)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Solve for x.

['] X
[←] [ALGEBRA] QUAD

```
1:                'X=18'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Getting General and Principal Solutions

HP 48 functions always return one result—the *principal* solution. For example, $\sqrt{4}$ always returns +2, and ASIN(.5) always returns 30 degrees or 0.524 radians.

However, when you solve an algebraic for a variable, there may be more than one solution—and you may want to know what they are. So the ISOL and QUAD commands normally return a *general* solution. A general solution represents the multiple solutions by including special variables that can take on multiple values:

- *s1* represents an arbitrary + or − sign (+1 or −1). Additional arbitrary signs in the result are indicated by *s2*, *s3*, … .
- *n1* represents an arbitrary integer—0, ±1, ±2, … . Additional arbitrary integers are represented by *n2*, *n3*, … .

To specify general or principal solutions:

- To get general solutions, press 1 [+/−] [→][MODES] [NXT] CF .
- To get principal solutions, press 1 [+/−] [→][MODES] [NXT] SF .

22 System flag −1 controls the type of solution returned by ISOL and QUAD. When you specify principal solutions, arbitrary signs are always chosen to be +1 and arbitrary integers are always chosen to be 0.

Example: Use ISOL to isolate x in the equation $y = \sin x^2$. Find both the principal and general solutions.

First, enter the equation. Then copy it. Set Radians mode, and set flag −1. Then, enter the variable to be isolated and get the principal solution.

['] Y [←][=] [SIN] X [y^x] 2
[ENTER] [ENTER]
[←][RAD] (if necessary)
1 [+/−] [→][MODES] [NXT] SF
['] X [←][ALGEBRA] ISOL

```
1:            'X=√ASIN(Y)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Clear flag −1. Then swap the copy of the original equation to level 1, enter the variable name, and get the general solution. The result contains the arbitrary sign *s1* and the arbitrary integer *n1*.

1 [+/−] [→][MODES] [NXT] CF
[←][SWAP]
['] X [←][ALGEBRA] ISOL

```
1: 'X=s1*√(ASIN(Y)*(-1
   )^n1+π*n1)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Press [←][RAD] to return to Degrees mode.

Showing Hidden Variables

Sometimes you may want to solve for a variable that's stored in another variable. To do this, you have to convert the algebraic so the hidden variable is visible.

Sometimes you may want to speed evaluation by converting an algebraic so all variables *except* certain ones are evaluated.

To convert an algebraic using partial evaluation:

- To show one hidden variable, enter the algebraic on the stack, enter the variable name (with ' delimiters), and press ⬅ ALGEBRA SHOW.
- To evaluate all variables except chosen ones, enter the algebraic on the stack, enter a list (with { } delimiters) containing the variable names to remain as names, and press ⬅ ALGEBRA SHOW.

22

Example: You want to solve 'A*B' for X, where A contains 'X+1'. Enter the expression 'A*B' and variable 'X' on the stack and press SHOW. You get '(X+1)*B', which you can solve for X.

Example: You want to draw a truth plot of 'X-Y+2*C>3*D', where C contains 7 and D contains 5. To save time, evaluate all variables except X and Y. Enter the expression 'X-Y+2*C>3*D' and the list { X Y } on the stack and press SHOW. You get 'X-Y+14>15', which you can plot.

Summary of Commands for Symbolic Solutions

The ALGEBRA Menu—Symbolic Solution Operations

Keys	Programmable Command	Description
⬅ ALGEBRA:		
ISOL	ISOL	For an algebraic in level 2, isolates the first occurrence of the variable in level 1.
QUAD	QUAD	Solves the quadratic in level 2 for the specified variable in level 1.
SHOW	SHOW	Shows the algebraic in level 2 with all implicit references to the variable in level 1 made explicit. For a list in level 1, evaluates all variables in the algebraic in level 2 that are not in the list.

22

Rearranging Terms

You can sometimes simplify algebraics by expanding subexpressions or collecting like terms. For example, if a variable occurs more than once in an algebraic, you may be able to simplify it so the variable occurs only once—letting you use ISOL to solve for the variable.

A *subexpression* consists of a function and its arguments. The function that defines a subexpression is called the *top-level* function for that subexpression—it's the function that's executed last. For example, in the expression 'A+B*C/D', the top level function for the subexpression 'B*C' is *, the top-level function for 'B*C/D' is /, and the top level function for 'A+B*C/D' is +.

Collecting Like Terms

To collect like terms in an algebraic:

- Enter the algebraic and press ⬏ ALGEBRA COLCT.

COLCT simplifies an algebraic by doing the following:

- Evaluates numerical subexpressions. For example, '1+2+LOG(10)' COLCT returns 4.
- Collects numerical terms. For example, '1+X+2' COLCT returns '3+X'.
- Orders factors (arguments of *) and combines like factors. For example, 'X^Z*Y*X^T*Y' COLCT returns 'X^(T+Z)*Y^2'.
- Orders summands (arguments of + or −) and combines like terms differing only in a coefficient. For example, 'X+X+Y+3*X' COLCT returns '5*X+Y'.

COLCT operates separately on the two sides of an equation, so like terms on the opposite sides of the equation are not combined.

22

Expanding Products and Powers

To expand products and powers in an algebraic:

- Enter the algebraic and press ⬏ ALGEBRA EXPA.

EXPAN rewrites an algebraic by doing the following:

- Distributes multiplication and division over addition. For example, 'A*(B+C)' EXPA returns 'A*B+A*C'.
- Expands powers over sums. For example, 'A^(B+C)' EXPA returns 'A^B*A^C'.
- Expands positive power integers. For example, 'X^5' EXPA returns 'X*X^4', and '(X+Y)^2' EXPA returns 'X^2+2*X*Y+Y^2'.

EXPAN doesn't carry out all possible expansions of an algebraic in a single execution. Instead, EXPAN works down through the subexpression hierarchy, stopping in each branch of the hierarchy when it finds a subexpression that it can expand. It first examines the top-level subexpression (the top level subexpression is the algebraic itself). If it's suitable for expansion, EXPAN expands it and stops—otherwise, EXPAN examines all of the second-level subexpressions. This process continues until an expansion occurs at some level—no lower levels are checked.

Example: Expand the expression `'A^(B*(C^2+D))'`.

Enter the expression.

['] A [y^x] [←] [( )] B [×] [←] [( )] C [y^x] 2 [+] D [ENTER]

```
1:        'A^(B*(C^2+D))'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

22 Expand the expression. The first expansion occurs at the second level—the subexpression `B*(C^2+D)` is expanded.

[←] [ALGEBRA] EXPA

```
1:        'A^(B*C^2+B*D)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Expand the expression again. The top-level function (the left ^) is expanded.

EXPA

```
1: 'A^(B*C^2)*A^(B*D)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Expand the expression again. The first expansion occurs at the third level—the subexpression `C^2` is expanded.

EXPA

```
1: 'A^(B*(C*C))*A^(B*D
   )'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

If you were to press EXPA, no further expansion would occur. Collect like terms.

COLCT

```
1:        'A^(B*C^2+B*D)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Example: The *EXCO* program on page 31-21 completely expands and collects an algebraic.

Summary of Commands for Collection and Expansion

The ALGEBRA Menu—Collection and Expansion Operations

Keys	Programmable Command	Description
(⬅)(ALGEBRA):		
COLCT	COLCT	Simplifies the algebraic in level 1 by collecting like terms.
EXPA	EXPAN	Rewrites the algebraic in level 1 by expanding subexpressions that contain products and powers.

22

Using the Rules Transformations

You can rearrange an algebraic in specific step-by-step stages, letting you get the result in the form you want. The Rules transformations are algebraic-rearrangement operations that are narrower in their scope than EXPAN and COLCT. The Rules transformations let you direct the path of an algebraic rearrangement.

To rearrange an algebraic in specific steps:

1. Put the algebraic in the EquationWriter application:
   - To enter a new algebraic, press (⬅)(EQUATION) and key it in.
   - To use an algebraic in level 1, press (▼).
   - To use an algebraic stored in a variable, enter the name (with ' delimiters) and press (➡)(▼).
2. Get the Selection environment:
   - From entry mode, press (◀).
   - From scrolling mode, press (⬅)(GRAPH) (◀).
3. Press (▲) (▼) (◀) (▶) to move the selection cursor to the *top-level function* for the subexpression you want to rearrange. (See below.)
4. Optional: Press EXPR at any time to show the associated subexpression—a highlight turns on or off.
5. Press RULES to get the RULES menu. (You can press (−) to return to the Selection menu.)

6. Press the menu key for the transformation you want. (Or just move the cursor to *not* do a transformation.)
7. Repeat step 6 for each transformation you want. (If you move the cursor, you have to go back to step 3.)
8. Press [ENTER] to save the transformed algebraic (or press [ATTN] to not save it).

In this section, the definition of *subexpression* in the previous section is expanded to include individual objects. For example, you can specify a name as the subexpression.

After you activate the Selection environment, you move the selection cursor—it specifies both an object in the algebraic and its corresponding subexpression.

22

Operations in the Selection Environment

Key	Description
RULES	Selects a menu of relevant rearrangement transformations for the specified subexpression.
EDIT	Returns the specified subexpression to the command line for editing. (See "Command-Line Editing" on page 16-17.)
EXPR	Highlights the specified subexpression.
SUB	Returns the specified subexpression to level 1 of the stack.
REPL	Replaces the specified subexpression with the algebraic in level 1 of the stack. (See "Replacing a Subexpression with an Algebraic Object" on page 16-22.)
EXIT	Exits the Selection environment, restoring the entry mode cursor at the end of the equation.
[▲] [▼] [◀] [▶]	Moves the selection cursor to the next object in the indicated direction. When prefixed with [↱], moves the selection cursor to the farthest object in the indicated direction.
[+/-]	Highlights the specified subexpression (just like EXPR), but is also active when the RULES menu is displayed.

The RULES menu may include transformations that aren't applicable to the specified subexpression—such menu keys produce a beep. After you execute a transformation, the selection cursor highlights the new top level object. The RULES menu is removed whenever you press any of the following keys: ◀ ▶ ▲ ▼, − (to return to the Selection menu), ENTER, or ATTN.

The tables on the next several pages describe the Rules transformations and show examples. However, the tables do *not* include all patterns for which transformations are applicable.

Note

The following tables include examples of transformations in the form

before → *after*

The before and after algebraics are shown in their *command-line form*—even though you execute Rules transformations in the *EquationWriter environment*. If you try an example, press ENTER to see the new expression in command-line form.

The RULES Menu—Universal Transformations

Key	Description
DNEG	Double-negate. A → --A
DINV	Double-invert. A → INV(INV(A))
*1	Multiply by 1. A → A*1 A+B/1 → A+B
^1	Raise to the power 1. A → A^1
/1	Divide by 1. A → A/1 A+B*1 → A+B

The RULES Menu—Universal Transformations (continued)

Key	Description
+1-1	Add 1 and subtract 1. A → A+1-1
COLCT	Collect. Executes a limited form of the COLCT command in the ALGEBRA menu. Works only on the subexpression defined by the specified object and leaves the coefficients of collected terms as sums or differences. (2+3)*X → 5*X 2*X+3*X → (2+3)*X

22

The RULES Menu—Moving Terms

Key	Description
←T	Move-term-left. Moves the nearest neighbor to the right of the specified function over the nearest neighbor to the left of the function. A+B+(C+D) → A+C+(B+D) A+B+(C+D) → A+B+(D+C) A+(B+C)*1+D → A+D+(B+C)*1 A*B=C*D → A*B/C=D
T→	Move-term-right. Moves the nearest neighbor to the left of the specified function over the nearest neighbor to the right of the function. A+B=(D+E) → A=-B+(D+E) A*B=(X+Y) → A=INV(B)*(X+Y)

←T and T→ are used to move a *term* over its "nearest neighbor" to the left or right. A term is an argument of + or − (a summand), an argument of ∗ or / (a factor), or an argument of =. Also, these two operations ignore parentheses—you can make them respect parentheses by executing *1 to make the parenthetical subexpression a term.

The RULES Menu—Building and Moving Parentheses

Key	Description
(( ))	Parenthesize-neighbors. Parenthesizes the nearest neighbors of + or ∗. Has no effect if the specified function is the first (or only) function in the expression, because these parentheses are already present, but hidden. A+B+C+D → A+(B+C)+D
(←	Expand-subexpression-left. Expands the subexpression associated with the specified function to include the next term to the left. Note that a matched pair of parentheses may disappear. A+B+(C+D)+E → A+(B+C+D)+E
→)	Expand-subexpression-right. Expands the subexpression associated with the specified function to include the next term to the right. A+(B+C)+D+E → A+(B+C+D)+E

22

The RULES Menu—Commuting, Associating, and Distributing

Key	Description
←→	Commute. Commutes the arguments of the specified function. A+B → B+A INV(A)*B → B/A
←A	Associate-left. A+(B+C) → A+B+C A*(B/C) → A*B/C A^(B*C) → A^B^C
A→	Associate-right. (A+B)+C → A+(B+C) (A*B)/C → A*(B/C) (A^B)^C → A^(B*C)
→()	Distribute-prefix-function. -(A+B) → -A-B INV(A/B) → INV(A)*B IM(A*B) → RE(A)*IM(B)+IM(A)*RE(B)

The RULES Menu—Commuting, Associating, and Distributing (continued)

Key	Description
←D	Distribute-left. (A+B)*C → A*C+B*C (A/B)^C → A^C/B^C
D→	Distribute-right. A*(B+C) → A*B+A*C A^(B-C) → A^B/A^C LN(A*B) → LN(A)+LN(B)
←M	Merge-factors-left. Merges arguments of +, −, *, and /, where the arguments have a common factor or a common single-argument function EXP, ALOG, LN, or LOG. For common factors, the ← indicates that the left-hand factors are common. Also merges sums where only one argument is a product. (A*B)+(A*C) → A*(B+C) EXP(A)*EXP(B) → EXP(A+B) A+A*B → A*(1+B)
M→	Merge-factors-right. Merges arguments of +, −, *, and /, where the arguments have a common factor. The → indicates that the right-hand factors are common. Also merges sums where only one argument is a product. (A*C)+(B*C) → (A+B)*C A*B+1*B → (A+1)*B
-()	Double-negate and distribute. Equivalent to DNEG followed by →() on the resulting inner negation. A+B → -(-A-B) LOG(INV(A)) → -LOG(A)
1/()	Double-invert and distribute. Equivalent to DINV followed by →() on the resulting inner inversion. A*B → INV(INV(A)/B) EXP(A) → INV(EXP(-A))

The RULES Menu—Rearranging Exponentials

Key	Description
L*	Replace log-of-power with product-of-log. LOG(A^B) → LOG(A)*B
L()	Replace product-of-log with log-of-power. LN(A)*B → LN(A^B)
E^	Replace power-product with power-of-power. ALOG(A*B) → ALOG(A)^B
E()	Replace power-of-power with power-product. EXP(A)^B → EXP(A*B)
→TRG	Replace exponential with trigonometric functions. (This example assumes Radians mode.) EXP(A) → COS(A/i)+SIN(A/i)*i

22

The RULES Menu—Adding Fractions

Key	Description
AF	Add fractions. Combines terms over a common denominator. (If the denominator is already common between two fractions, use M→ .) A+(B/C) → (A*C+B)/C (A/B)-C → (A-B*C)/B

The RULES Menu—Expanding Trigonometric Functions

Key	Description
→DEF	Expand-trigonometric-definition. Replaces trigonometric, hyperbolic, inverse trigonometric, and inverse hyperbolic functions with their definitions in terms of EXP and LN. (These examples assume Radians mode.) COS(X) → (EXP(X*i)+EXP(-(X*i)))/2 ASINH(U) → -LN(√(1+U^2)-U)
TRG*	Expand as product-of-trigonometric-functions. Expands trigonometric functions of sums and differences. SIN(X+Y) → SIN(X)*COS(Y)+COS(X)*SIN(Y)

The RULES Menu—Automatic Multiple Execution

Key	Description
↱ D→	Multiple-distribute-right.
↱ ←D	Multiple-distribute-left. (A+B+C)*D → A*D+B*D+C*D
↱ A→	Multiple-associate-right.
↱ ←A	Multiple-associate-left. A+(B+(C+D)) → A+B+C+D
↱ M→	Multiple-merge-factors-right. A*B+C*B+D*B → (A+C+D)*B
↱ ←M	Multiple-merge-factors-left.
↱ T→	Multiple-move-term-right. A+B+C+D=E → B+C+D=E−A
↱ ←T	Multiple-move-term-left.
↱ →)	Multiple-expand-subexpression right. A+(B+C)+D+E → A+(B+C+D+E)
↱ (←	Multiple-expand-subexpression-left.

Prefixing the previous transformation keys with ↱ causes that transformation to execute repeatedly until no further change occurs.

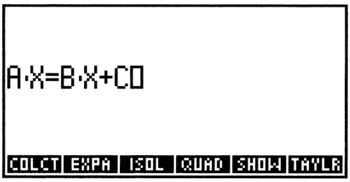
Example: Solve for the variable x in the equation

$$ax = bx + c$$

Do this by rearranging the equation so x appears only once, then using ISOL.

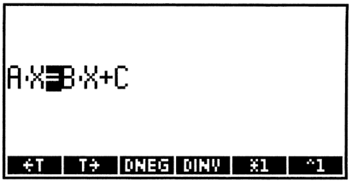
Select the EquationWriter application and key in the expression.

↰ EQUATION
A × X ↰ =
B × X + C



Activate the Selection environment. Then move the selection cursor to the = sign and get the RULES menu.

[◀]
[◀] (5 times)
RULES



Move the term B·X to the left side of the = sign.

←T

22

Merge the two terms on the left side of the = sign.

M→

Now that x occurs only once in the equation, put the equation on the stack and isolate x.

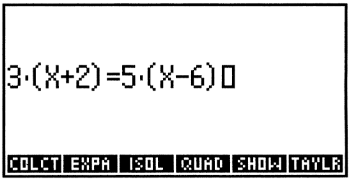
[ENTER]
[↰] [ALGEBRA]
['] X ISOL

Example: Solve for x in the equation

$$3(x+2) = 5(x-6)$$

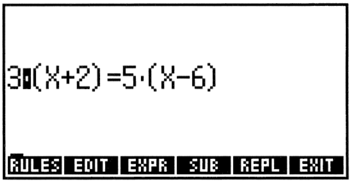
Select the EquationWriter application and key in the equation.

[↰] [EQUATION]
3 [↰] [()] X [+] 2 [▶]
[↰] [=]
5 [↰] [()] X [−] 6 [▶]



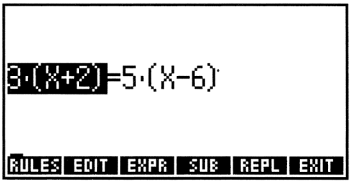
Activate the Selection environment and move the selection cursor to the · sign on the left side of the equation.

[◀]
[↱] [◀] [▶]



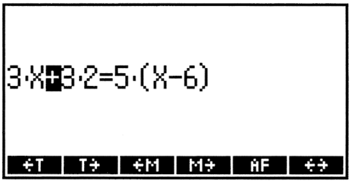
Highlight the subexpression defined by ·. This shows you what part of the equation will be affected by the rearrangement.

EXPR



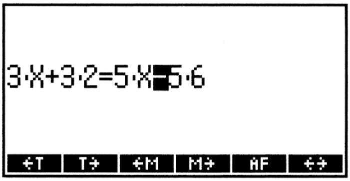
Select the RULES menu for this subexpression and distribute the 3 over (X+2).

RULES D→



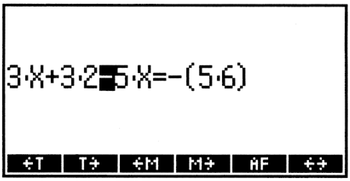
Move the selection cursor to the · on the right side of the equation and distribute again.

[▶] (6 times)
RULES D→



Move the cursor to the = sign and then move the term 5·X to the left side of the equation.

[◀] (4 times)
RULES ←T



Now that both terms in X are on the same side of the equation, return the equation to the stack and collect like terms.

(ENTER)
(↰)(ALGEBRA) COLCT

Now solve for X.

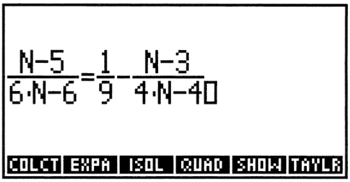
(') X ISOL

Example: Solve for n in the equation

$$\frac{n-5}{6n-6} = \frac{1}{9} - \frac{n-3}{4n-4}$$

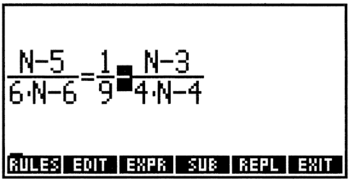
Key in the equation.

(↰)(EQUATION)
(▲) N (−) 5 (▶) 6N (−) 6 (▶)
(↰)(=) 1 (÷) 9 (▶) (−)
(▲) N (−) 3 (▶) 4N (−) 4



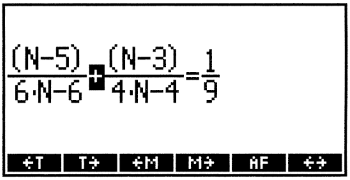
Activate the Selection environment. Move the cursor to the − sign between the two right-hand terms.

(◀)
(◀) (as required)



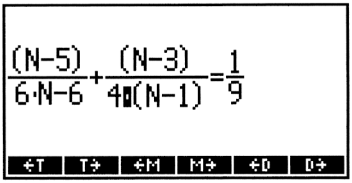
Move the rightmost term to the left side of the equation.

RULES ←T ←T



Move the cursor to the − sign in the denominator `4*N-4` and merge factors left.

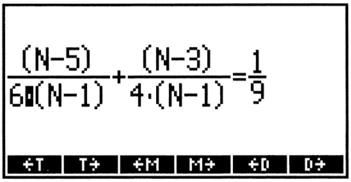
▶ ▼
RULES ←M



Move the cursor to the − sign in the denominator of the first term on the left side and merge factors left.

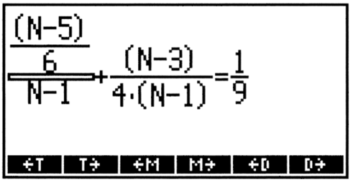
22

◀ ◀ ◀ ▼
RULES ←M



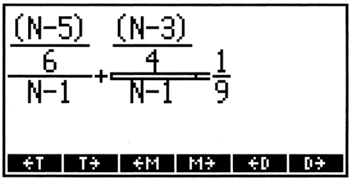
Move the cursor to the divide bar of that term and associate left.

▲
RULES NXT ←A



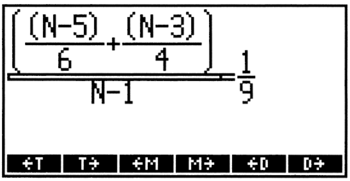
Move the cursor to the divide bar of the second term and associate left.

▶ ▶
RULES NXT ←A



Move the cursor to the + sign between the two terms and merge factors right.

◀
RULES M→



Move the cursor to the = sign and move term right.

(▶)
RULES T→

$$\frac{(N-5)}{6}+\frac{(N-3)}{4}=(N-1)\blacksquare\left(\frac{1}{9}\right)$$

```
←T | T→ | ←M | M→ | ←D | D→
```

Put the equation on the stack. Set the display mode to 1 Fix (to see the result of the subsequent COLCT operation more easily). Expand terms and then collect like terms.

(ENTER)
(↰)(MODES) 1 FIX
(↰)(ALGEBRA)
EXPA COLCT

```
1: '-1.6+0.4*N=-0.1+
    0.1*N'
COLCT | EXPA | ISOL | QUAD | SHOW | TAYLR
```

22

Solve for N.

(') N QUAD

```
1:                   'N=4.8'
COLCT | EXPA | ISOL | QUAD | SHOW | TAYLR
```

Press (↰)(MODES) STD to return to Standard display mode.

Making User-Defined Transformations

If the built-in set of Rules transformations do not rearrange an algebraic in the form you desire, you can make your own transformations. By making a "custom" transformation, you can replace occurrences of a pattern with a new pattern. The pattern can be specific—or it can contain "wildcards" that match any subexpression and that you can reinsert in the replacement. And you're informed whether or not a replacement was made.

You can also make conditional transformations—the transformation occurs or not depending on a condition you specify.

To make a custom transformation on an algebraic:

1. Enter the algebraic on the stack.
2. Enter a list (with { } delimiters) that specifies the transformation:

- To make an *unconditional* transformation, include the search and replacement patterns (with ' delimiters).
- To make a *conditional* transformation, include the search and replacement patterns *and the conditional expression* (with ' delimiters).

3. Press (←)(ALGEBRA) (NXT) ↑MAT or ↓MAT .

The list specifying the transformation has one of these forms:

{ '*search*' '*replace*' }
{ '*search*' '*replace*' '*conditional*' }

22 The ↑MATCH and ↓MATCH commands search for the specified pattern and replace all occurrences of that pattern with the replacement pattern. For a conditional transformation, the replacement occurs only if the conditional expression evaluates to a nonzero value (true).

If a replacement *is* made, the new expression is returned to level 2—and 1 (true) is returned to level 1. If a replacement *is not* made, the original expression is returned to level 2—and 0 (false) is returned to level 1.

↑MATCH starts its search at the lowest level subexpression and works up—this works well for simplification. ↓MATCH starts with the complete algebraic and works down—this works well for expansion. Replacement stops at the end of the first level in which a replacement occurs. You can repeat the transformation to change other levels.

For generalized transformations, the search pattern can contain "wildcard" names that match *any* subexpressions. When the replacement pattern is inserted, each wildcard name is replaced by the matching subexpression from the search pattern. A wildcard name consists of an & character ((α) (←) (ENTER)) and a valid variable name, such as &A.

Example: An extension of the half-angle formula for sine is

$$\sin(2z) = 2\sin(z)\cos(z)$$

There is no Rules transformation for this formula, so create a user-defined transformation list for this transformation. Then use it to transform the expression 'SIN(2*(X+1))'.

Create the transformation list. (To key in &, press (α) (↰)(ENTER).)

(↰)({ }) (') (SIN) 2 (×) &W (▶) (▶)
(') 2 (×) (SIN) &W (▶) (×) (COS) &W
(ENTER)

```
1: { 'SIN(2*&W)' '2*
   SIN(&W)*COS(&W)' }
STD ■ | FIX | SCI | ENG | SYM ■ | BEEP ■
```

Store the list in variable *HALF*. Then enter the expression to transform.

(') HALF (STO)
(') (SIN) 2 (×) (↰)(( )) X (+) 1
(ENTER)

```
1:         'SIN(2*(X+1))'
STD ■ | FIX | SCI | ENG | SYM ■ | BEEP ■
```

Recall the transformation list from the VAR menu, then use ↓MATCH to transform the expression. The value in level 1 shows a replacement occurred.

(VAR) HALF
(↰)(ALGEBRA) (NXT) ↓MAT

```
2: '2*SIN(X+1)*COS(X+...
1:                     1
↑MAT | ↓MAT | | | APPLY | QUOT | →Qπ
```

Drop level 1 to see the transformed expression.

(◆)

```
1: '2*SIN(X+1)*COS(X+1
   )'
↑MAT | ↓MAT | | | APPLY | QUOT | →Qπ
```

As an alternative, you can include the list and the ↓MATCH command in a program—then you can make the transformation in one step.

Using the | (Where) Function

The | function ((↰)(ALGEBRA) (NXT) |)—read as "where" or "evaluated at"—binds numeric values to variables that occur in a partially evaluated algebraic. It provides a way to do stepwise evaluation of integrals and user-defined functions—it provides substitution information about names, even if the names no longer exist, as can occur with local variables.

You can also use | to evaluate an algebraic for specific variable values.

To evaluate an algebraic for specific variable values:

1. Enter the algebraic on the stack.
2. Enter a list (with { } delimiters) that contains each variable name followed by the value to substitute. (See below.)
3. Press (←)(ALGEBRA) | .

The list of names and values should have the form

{ $name_1$ $expr_1$ ... $name_n$ $expr_n$ }

where *expr* can be a number or a symbolic expression. If a variable named in the list currently exists, its contents are *not* changed by | (where).

22

Example: The evaluation of an integral returns a symbolic result of the form

'*expr*|(*var*=*upper-limit*)-(*expr*|(*var*=*lower-limit*))'

as described in the next chapter, "Calculus." Here *expr* is the integrated expression, still in symbolic form—and *var* is the variable of integration. Press (EVAL) to substitute the limits of integration.

Example: Consider the user-defined function *DRV* created by entering 'DRV(X)=∂X(X^2)' and pressing (←)(DEF). Enter 'DRV(2)' and press (EVAL) to return the partially evaluated result

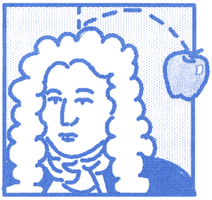
'∂X(X)*2*X^(2-1)|(X=2)'

X is a local variable and exists only while the user-defined function *DRV* is being executed—so the | function supplies substitution information for the terminated local variable. Press (EVAL) again to get the final answer 4.

Example: Evaluate $A + B$, where $A = C + D$ and $B = 7$. Enter 'A+B', then enter { A 'C+D' B 7 }. Press | to get the expression 'C+D+7'.

23

Calculus



You can use the HP 48 calculus commands to differentiate expressions, calculate series summations, derive Taylor's polynomials, and perform symbolic and numeric integration.

Differentiating Expressions

You can differentiate a symbolic expression either one step at a time, so you can see the substitutions—or completely in one step, so you can go right to the final result. If your expression contains only analytic functions (those labeled with "A" in appendix G), you get an explicit derivative.

Differentiating Step-by-Step

To differentiate an expression step-by-step:

1. Enter a symbolic expression (with ' delimiters) for the ∂ function with the expression to differentiate as the argument.
2. To perform each step, press EVAL.

When you use ∂ in *algebraic* syntax, it differentiates the expression step-by-step. In algebraic syntax, ∂ has the command-line form

'∂*var*(*expression*)'

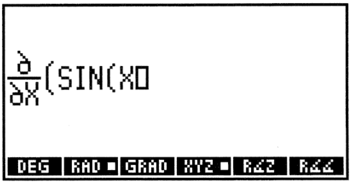
where *var* is the variable of differentiation and *expression* is the expression you're differentiating. You can enter the symbolic expression in the EquationWriter application—see under "Entering Equations" on page 16-8.

Example: Calculate the derivative of $\sin x$ using a symbolic expression.

(This example assumes that variable X does not exist in the current directory—you can press ['] X [←][PURGE].)

Select Radians mode, and key in the derivative of the expression, using the EquationWriter application.

[←][MODES] [NXT] [NXT]
RAD (if necessary)
[←][EQUATION]
[→][∂] X [▶] [SIN] X



23

Put the expression on the stack, and evaluate the two steps to get the final result.

[ENTER]
[EVAL] [EVAL]

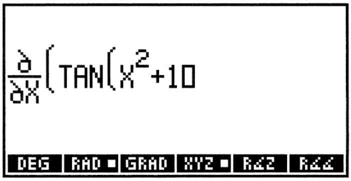
Example: Calculate step-by-step the expression

$$\frac{d}{dx}\tan(x^2+1)$$

(This example assumes that variable X does not exist in the current directory—you can press ['] X [←][PURGE].)

Set the angle mode to Radians. Select the EquationWriter application and key in the derivative.

[←][RAD] (if necessary)
[←][EQUATION]
[→][∂] X [▶]
[TAN] X [y^x] 2 [▶] [+] 1



Evaluate the expression.

[EVAL]

The result still contains a derivative, illustrating the chain rule of differentiation:

$$\frac{d}{dx}\tan(x^2+1) = \frac{d}{d(x^2+1)}\tan(x^2+1) \times \frac{d}{dx}(x^2+1)$$
$$= (1+\tan^2(x^2+1)) \times \frac{d}{dx}(x^2+1)$$

The derivative of the tangent function has already been evaluated. Evaluate the next step, the derivative of x^2+1.

(EVAL)

```
1: '(1+TAN(X^2+1)^2)*∂
   X(X^2)'
 DEG | RAD ■| GRAD | XYZ ■| R∠Z | R∠∠
```

23

The operation evaluates the derivative of the sum

$$\frac{d}{dx}(x^2+1) = \frac{d}{dx}x^2 + \frac{d}{dx}1$$

The derivative of 1 is 0, so the term disappears. Evaluate the next step, the derivative of x^2.

(EVAL)

```
1: '(1+TAN(X^2+1)^2)*(
   ∂X(X)*2*X^(2-1))'
 DEG | RAD ■| GRAD | XYZ ■| R∠Z | R∠∠
```

The operation reflects the chain rule:

$$\frac{d}{dx}x^2 = \frac{d}{dx}(x)^2 \times \frac{d}{dx}(x)$$

The derivative of x^2 has been evaluated. Evaluate the final step.

(EVAL)

```
1: '(1+TAN(X^2+1)^2)*(
   2*X)'
 DEG | RAD ■| GRAD | XYZ ■| R∠Z | R∠∠
```

Differentiating Completely

To differentiate an expression completely in one step:

1. Enter the expression you want to differentiate (with ' delimiters).
2. Enter the variable of differentiation (with ' delimiters).
3. Press (↱)(∂).

Example: Calculate in one step the expression

$$\frac{d}{dx}\tan(x^2+1)$$

(This example assumes that variable X does not exist in the current directory—you can press ['] X [←][PURGE].)

Enter the expression. Enter the variable of differentiation.

['] [TAN] X [y^x] 2 [+] 1 [ENTER]
['] X [ENTER]

```
2:           'TAN(X^2+1)'
1:                    'X'
 DEG  RAD■ GRAD  XYZ■  R∠Z  R∠∠
```

Differentiate the expression.

[→][∂]

```
1: '(1+TAN(X^2+1)^2)*(
   2*X)'
 DEG  RAD■ GRAD  XYZ■  R∠Z  R∠∠
```

Differentiating User-Defined Functions

You can differentiate user-defined functions. See "Differentiating a User-Defined Function" on page 10-3.

Creating User-Defined Derivatives

If you execute ∂ for a function that has no built-in derivative, ∂ returns a *new* function whose name is *der* followed by the original function name. The new function has arguments that are the arguments of the original function, plus the arguments' derivatives. (You can differentiate further by creating a user-defined function to represent the new derivative function.)

If you execute ∂ for a formal user function (a name followed by arguments in parentheses, for which no user-defined function exists in user memory), ∂ returns a formal derivative whose name is *der* followed by the original user function name, plus the arguments and their derivatives.

Example: The HP 48 definition of % does not include a derivative. If you enter `'∂Z(%(X,Y))'` and press [EVAL], you get

```
'der%(X,Y,∂Z(X),∂Z(Y))'
```

Each argument of the % function results in two arguments for the der% function—X results in X and ∂Z(X), and Y results in Y and ∂Z(Y).

To define the derivative function for %, you can enter 'der%(x,y,dx,dy)=(x*dy+y*dx)/100' and press (⬅)(DEF).

Now you can obtain the derivative of '%(X,2*X)' by entering the expression and the variable 'X', then pressing (➡)(∂) (⬅)(ALGEBRA) COLCT. The result is '.04*X'.

Example: Enter the derivative of a formal user function, ∂x('f(x1,x2,x3)'). Then evaluate it by press (EVAL). The result is

```
'derf(x1,x2,x3,∂x(x1),∂x(x2),∂x(x3))'
```

23

Summing Finite Series

You can calculate the value of a finite series. You can also use this capability to explore whether or not an infinite series converges.

To calculate a summation using algebraic syntax:

1. Enter the symbolic expression for the Σ function with the index, limits, and summand as arguments.
2. Press (EVAL).

When you use Σ in *algebraic* syntax, it has the command-line form

'Σ(*index*=*initial*, *final*, *summand*)'

where *index* is the index variable name, *initial* and *final* are the first and last values of the index variable, and *summand* is an expression representing the terms being summed. You can enter the summation in the EquationWriter application—see under "Entering Equations" on page 16-9.

To calculate a summation using stack syntax:

1. Enter the name of the index variable you'll use in the summand expression (with ' delimiters).
2. Enter the initial value of the index.

3. Enter the final value of the index.
4. Enter an expression for the summand (with ' delimiters).
5. Press (↱)(Σ).

Example: Calculate

$$\sum_{n=1}^{50} \frac{(-1)^n n}{2^n}$$

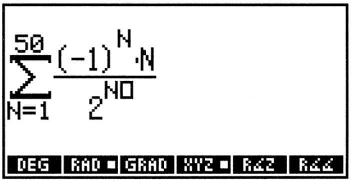
Select the EquationWriter application and key in the Σ function, the summation index, the initial value, and the final value.

23

(↰)(EQUATION)
(↱)(Σ)
N (▶)
1 (▶)
50 (▶)

Key in the summand.

(↰)(( )) (−) 1 (▶)
(y^x) N (▶) N
(÷) 2 (y^x) N



Calculate the sum.

(EVAL)

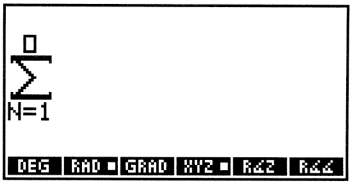
Example: For the infinite geometric series

$$\sum_{n=1}^{\infty} r^{n-1}$$

see whether the series converges or diverges for $r = 0.5$.

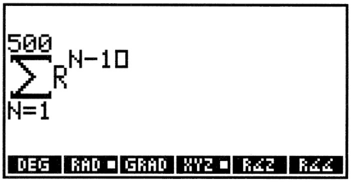
Select the EquationWriter application, and key in the summation function, the index, and its initial value.

[←] [EQUATION]
[→] [Σ]
N [▶] 1 [▶]



Key in a final value for the summation index. Because the HP 48 can't represent infinity, use a large number, like 500. Then key in the summand.

500 [▶]
R [y^x] N [−] 1



Enter the expression. Make two extra copies to use in this and the next example. Store the value 0.5 in R and calculate the sum. (The calculation takes several seconds.)

[ENTER] [ENTER] [ENTER]
.5 ['] R [STO]
[EVAL]

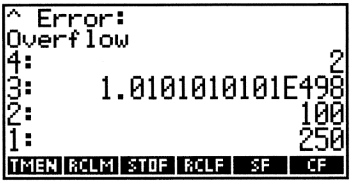
Swap the expression into level 1 and change the final value of the index to 1000, then calculate the sum again. (The calculation takes several seconds.) The calculations suggest that the series converges to 2.

[▶] [←] [EDIT]
[▶] (7 times) [DEL] 10 [ENTER]
[EVAL]

Example: Evaluate the series from the previous example to see whether it converges or diverges for $r = 100$. (This example assumes the summation expression remains from the previous example.)

Set system flag −21 so that numbers larger than MAXR (*maximum real number*) cause an overflow error. Then, swap the remaining copy of the expression into level 1, store 100 in R, and calculate the sum. An overflow error occurs, suggesting that the series diverges.

21 [+/-] [→] [MODES] [NXT] SF
[▲]
[▲] [▲] ROLL [ATTN]
100 ['] R [STO]
[EVAL]



23 Press 21 [+/-] CF to not generate overflow errors.

Deriving Taylor's Polynomial Approximations

For any mathematical function represented by a symbolic expression, you can compute a Taylor's polynomial approximation about $x = 0$, sometimes called a Maclaurin series. You can also specify the order of the polynomial.

To derive the Taylor's polynomial approximation about x = 0:

1. Enter an expression for the function being approximated (with ' delimiters).
2. Enter the name of the variable for the polynomial (with ' delimiters).
3. Enter the order of the polynomial (the maximum power for the variable).
4. Press [←] [ALGEBRA] TAYLR.

The TAYLR command can't be used in algebraic syntax.

Example: Calculate the 3rd-order Taylor's polynomial about $x = 0$ for

$$\frac{1}{\sqrt{1 + x^3}}$$

(This example assumes that variable X does not exist in the current directory—you can press ['] X [←] [PURGE].)

Enter the expression, the polynomial variable, and the order of the polynomial.

['] 1 [÷] [√x] [⬅][( )] 1 [+] X [y^x] 3
[ENTER]
['] X [ENTER]
3 [ENTER]

```
3:          '1/√(1+X^3)'
2:                   'X'
1:                     3
TMEN RCLM STOF RCLF  SF   CF
```

Derive the approximation. (The calculation takes several seconds.)

[⬅][ALGEBRA] TAYLR

```
1:           '1-3/3!*X^3'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Evaluate to complete the calculation.

23

[EVAL]

```
1:             '1-.5*X^3'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Example: Calculate the 5th-order Taylor's polynomial approximation about $x = 0$ for $\sin x$.

(This example assumes that variable X does not exist in the current directory—you can press ['] X [⬅][PURGE].)

Select Radians mode. Enter the expression, the polynomial variable (X), and the order of the polynomial. Then find the Taylor's polynomial.

[⬅][RAD] (if necessary)
['] [SIN] X [ENTER]
['] X [ENTER]
5 [⬅][ALGEBRA] TAYLR

```
1:  'X-1/3!*X^3+1/5!*X^
     5'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Evaluate the expression for $X = 0.5$.

.5 ['] X [STO]
[EVAL]

```
1:          .479427083334
COLCT EXPA ISOL QUAD SHOW TAYLR
```

For comparison, .5 [SIN] returns .4794255.... The approximation is accurate to five decimal palces.

To derive the Taylor's polynomial approximation about x = a:

1. Purge a dummy variable Y.
2. Store $Y + A$ in the polynomial variable X.

3. Enter an expression for the function being approximated.
4. Press EVAL to change the variable from X to Y.
5. Enter the name of the variable Y.
6. Enter the order of the polynomial.
7. Press ⬅ ALGEBRA TAYLR.
8. Purge variable X.
9. Store $X - A$ in Y.
10. Press EVAL to change the variable from Y to X.

For A in the previous steps, you can use a number or a variable.

TAYLR always evaluates the function and its derivatives at zero. If you're interested in the behavior of a function in a region away from zero, the Taylor's polynomial will be more useful if you translate the point of evaluation to that region, as described above. Also, if the function has no derivative at zero, its Taylor's polynomial will be meaningless unless you translate the point of evaluation away from zero.

Integrating Expressions

You can calculate *symbolic* integrals for expressions with known antiderivatives (indefinite integrals). You can also estimate the *numeric* value of those and other integrals.

Doing Symbolic Integration

Symbolic integration means calculating an integral by finding a known antiderivative and then substituting specified limits of integration. The result is a symbolic expression.

The HP 48 can integrate the following patterns:

- All built-in functions whose antiderivaties contain only built-in functions (and whose arguments are linear). See the analytic functions, labeled with "A" in appendix G. For example, 'SIN(X)' → 'COS(X)'.
- Sums, differences, negations, and other selected patterns of such functions. For example, 'SIN(X)-COS(X)' →

'-SIN(X)-COS(X)', and '1/(COS(X)*SIN(X))' → 'LN(TAN(X))'.

- Derivatives of all built-in functions. For example, 'INV(1+X^2)' → 'ATAN(X)'.
- Polynomials whose base term is linear. For example, '(X-3)^3+6' → '6*X+(X-3)^4/4'.

To find a symbolic integral using algebraic syntax:

1. Enter the symbolic expression for the ∫ function with the limits, integrand, and variable of integration as arguments.
2. Press (EVAL) (EVAL).

23

When you use ∫ in *algebraic* syntax, it has the command-line form

'∫(*lower*, *upper*, *integrand*, *var*)'

where *lower* and *upper* are the limits of integration, *integrand* is the expression being integrated, and *var* is the variable of integration. You can enter the integration in the EquationWriter application—see under "Entering Equations" on page 16-8.

To find a symbolic integral using stack syntax:

1. In the MODES menu, make sure SYM■ is displayed.
2. Enter the lower limit of integration.
3. Enter the upper limit of integration.
4. Enter the integrand, the expression you want to integrate (with ' delimiters).
5. Enter the variable of integration (with ' delimiters).
6. Press (↱)(∫).
7. Press (EVAL).

The result of symbolic integration indicates the success of the integration:

- If the result is a *closed-form* expression—if there is no ∫ sign in the result—the symbolic integration was successful.
- If the result still contains ∫, you can try rearranging the expression and evaluating again. If rearranging fails to produce a closed form result, you can estimate the answer with numeric integration, described under "Doing Numeric Integration" on page 23-14.

Before you press the final [EVAL], a closed-form result of $\int$ has the form

'*result*|(*var*=*upper*)−(*result*|(*var*=*lower*))'

where *result* is the closed-form integral, *var* is the variable of integration, and *upper* and *lower* are the limits. (The | (where) function is discussed under "Using the | (Where) Function" on page 22-25.

Example: Calculate

$$\int_0^y (x^2 + 1)dx$$

(This example assumes variable Y does not exist in the current directory—you can press ['] Y [←][PURGE].)

Select the EquationWriter application and key in the $\int$ function, its limits, the integrand, and the variable of integration.

[←][EQUATION]
[→][∫]
0 [▶] Y [▶]
X [y^x] 2 [▶] [+] 1
[▶] X

```
⌠Y 2
⎮  X +1 dX
⌡0
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Evaluate the expression.

[EVAL]

```
RAD
{ HOME }
1: '1*X+X^(2+1)/((2+1)
   *∂X(X))|(X=Y)-(1*X+
   X^(2+1)/((2+1)*∂X(X
   ))|(X=0))'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

The result is closed form. Now evaluate again to substitute the limits into the variable of integration.

[EVAL]

```
1:              'Y+Y^3/3'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Example: Calculate

$$\int_0^y (x^2 + 1)^2 dx$$

(This example assumes variable Y does not exist in the current directory—you can press ['] Y [⬅] [PURGE].)

Select the EquationWriter application, then key in the integral sign, the limits, the integrand, and the variable of integration.

[⬅] [EQUATION]
[➡] [∫]
0 [▶] Y [▶]
[⬅] [( )] X [y^x] 2 [▶] [+] 1
[▶] [y^x] 2 [▶] [▶] X

$$\int_0^Y \left(X^2+1\right)^2 \, dX$$

```
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Try to calculate the integral. The operation isn't successful because the term (X^2+1) isn't linear.

[EVAL]

```
1: '∫(0,Y,(X^2+1)^2,X)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Rearrange the expression by expanding and collecting.

[⬅] [ALGEBRA]
EXPA EXPA EXPA
COLCT

```
1: '∫(0,Y,1+X^4+2*X^2,
   X)'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Now evaluate the rearranged expression.

[EVAL]

```
RAD
{ HOME }
1: '2*(X^(2+1)/((2+1)*
   ∂X(X))+X^(4+1)/((4
   +1)*∂X(X))+1*X|(X=Y
   )-(2*(X^(2+1)/((2+1
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Complete the integration by evaluating the | (where) functions.

[EVAL]

```
1: '2*(Y^3/3)+Y^5/5+Y'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

To symbolically integrate an expression that's not integrable:

1. Derive a Taylor's polynomial approximation to the integrand.
2. Find the symbolic integral of the polynomial.

Not all expressions are directly integrable on the HP 48—see "How the HP 48 Does Symbolic Integration" on page 23-18. You may be able to use the TAYLR command to approximate the integrand.

Example: Calculate

$$\int_0^y e^{x^2}\,dx$$

The integrand is not integrable by any of the methods described so far in this chapter. Calculate a 4th-order Taylor's polynomial for this expression and integrate the polynomial.

(This example assumes that variables X and Y do not exist in the current directory—you can press [←][{ }] X [SPC] Y [ENTER] [←][PURGE].)

23 Enter the expression. Enter the series variable and the order of the polynomial, then find the Taylor's polynomial. (The calculation takes several seconds.) Then evaluate the result.

['] [←][eˣ] X [yˣ] 2 [ENTER]
['] X [ENTER] 4 [←][ALGEBRA] TAYLR
[EVAL]

```
1:         '1+X^2+.5*X^4'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Use stack syntax to calculate the integral: Enter the lower and upper limits and move the integrand to level 1.

0 ['] Y [ENTER]
[▲]
[▲] [▲] ROLL [ATTN]

```
3:                      0
2:                    'Y'
1:         '1+X^2+.5*X^4'
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Enter the variable of integration, integrate the expression, and evaluate the result. This approximation is less accurate for Y not near 0.

['] X [ENTER]
[→][∫]
[EVAL]

```
1: .5*(Y^5/5)+Y^3/3+Y
COLCT EXPA ISOL QUAD SHOW TAYLR
```

Doing Numeric Integration

Numeric integration lets you approximate a definite integral—even when symbolic integration can't generate a closed-form result. Numeric integration employs an iterative numeric procedure to obtain the approximation.

To find the value of an integral using algebraic syntax:

1. Specify the accuracy factor for the integrand:
   - For an accuracy factor of 10^{-n}, press n ⬅ MODES FIX .
   - For an accuracy factor of 10^{-11}, press ⬅ MODES STD .
2. Enter the symbolic expression for the $\int$ function with the limits, integrand, and variable of integration as arguments.
3. Press ➡ →NUM.

When you use $\int$ in *algebraic* syntax, it has the command-line form

'∫(*lower*, *upper*, *integrand*, *var*)'

To find the value of an integral using stack syntax:

23

1. Specify the accuracy factor for the integrand:
   - For an accuracy factor of 10^{-n}, press n ⬅ MODES FIX .
   - For an accuracy factor of 10^{-11}, press ⬅ MODES STD .
2. Enter the lower limit of integration.
3. Enter the upper limit of integration.
4. Enter the integrand, the expression you want to integrate (with ' delimiters).
5. Enter the variable of integration (with ' delimiters).
6. Press ➡ ∫.
7. Press ➡ →NUM.

The display format specifies the accuracy factor. The accuracy factor determines the acceptable tolerance between the final iterations of the numeric procedure. Except in rare cases, this factor is the percent error in the result. For example, to specify an accuracy factor of 0.0001 (0.01%), press 4 FIX . Generally, the smaller the tolerance, the longer the calculation.

To check the uncertainty of the numeric result:

- Press VAR IERR .

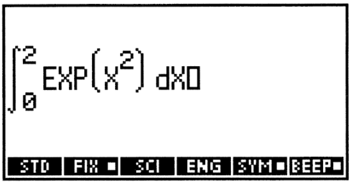
If the uncertainty of integration *IERR* is too large, the integral is unreliable. If *IERR* is −1, the integral didn't converge.

Example: Use numeric integration (accuracy factor of 0.0001) to calculate

$$\int_0^2 e^{x^2}\,dx$$

Specify the accuracy factor. Select the EquationWriter application, key in the integral function, limits of integration, integrand, and variable of integration.

[⬅] [MODES] 4 FIX
[⬅] [EQUATION]
[➡] [∫] 0 [▶] 2 [▶]
[⬅] [e^x] X [y^x] 2 [▶] [▶] [▶] X



Calculate the numeric approximation. (The calculation takes several seconds.)

[➡] [→NUM]

Press STD to return to Standard display format.

(In the previous example, you used a Taylor's polynomial to approximate the same integral symbolically for Y near 0. Evaluating that integral for $Y = 2$ (*not* near 0) returns the inaccurate result 7.87.)

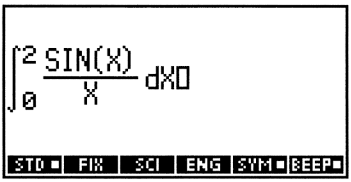
Example: Calculate Si(2 degrees), where Si(t) is the *sine* integral (sometimes used in communications theory)

$$\mathrm{Si}(t) = \int_0^t \frac{\sin x}{x} dx$$

Because the integrand $(\sin x)/x$ is a purely mathematical expression containing no empirically-derived constants, the only constraint on its accuracy is the round-off error introduced by the calculator. It is, therefore, at least analytically reasonable to specify an accuracy factor of 1×10^{-11}.

Set the angle mode to Degrees. Set the display mode to Standard. Key in the integral function, limits of integration, integrand, and variable of integration.

[⬅] [RAD] (if necessary)
[⬅] [MODES] STD (if necessary)
[⬅] [EQUATION]
[➡] [∫] 0 [▶] 2 [▶]
[SIN] X [▶] [÷] X [▶] [▶] X



Put the integral on the stack and make a copy to use later. Calculate the integral.

(ENTER) (ENTER)
(↱)(→NUM)

```
1:     3.49042222032E-2
 STD ■ | FIX | SCI | ENG | SYM ■ | BEEP ■
```

Check the uncertainty of integration. The uncertainty is significant only with respect to the last digit of the integral.

(VAR) IERR

```
2:     3.49042222032E-2
1:      3.490422336E-13
 IERR | HALF | C | S1 | GAS | ΣPAR
```

Repeat the calculation for a larger tolerance—an accuracy factor of 0.001. Set the display mode to 3 Fix, move the expression to level 1, then evaluate it. 23

(↰)(MODES) 3 FIX
(▲)(▲)(▲) ROLL (ATTN)
(↱)(→NUM)

```
3:                0.035
2:            3.490E-13
1:                0.035
 STD | FIX ■ | SCI | ENG | SYM ■ | BEEP ■
```

Check the new uncertainty of integration. The second uncertainty is much larger—but it's still relatively small compared to the value of the integral—and the calculation is faster.

(VAR) IERR

```
4:                0.035
3:            3.490E-13
2:                0.035
1:             3.490E-5
 IERR | HALF | C | S1 | GAS | ΣPAR
```

Press (↰)(MODES) STD to set Standard display mode.

More about Integration

This section gives you some details about symbolic and numeric integration.

How the HP 48 Does Symbolic Integration

The HP 48 does symbolic integration by *pattern matching*. The HP 48 can integrate:

23

- All the built-in functions whose antiderivatives are expressible in terms of other built-in functions—for example, SIN is integrable 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 are expressible in terms of other built-in functions—for example, `'SIN(X)-COS(X)'`.
- Derivatives of all the built-in functions—for example, `'INV(1+X^2)'` is integrable 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'` is integrable since `X` is a linear term. `'(X^2-6)^3+(X^2-6)^2'` is not integrable since `X^2-6` is not linear.
- Selected patterns composed of functions whose antiderivatives are expressible in terms of other built-in functions—for example, `'1/(COS(X)*SIN(X))'` returns `'LN(TAN(X))'`.

The Accuracy Factor and the Uncertainty of Integration

Numeric integration calculates the integral of a function $f(x)$ by computing a weighted average of the function's values at many values of x (sample points) within the interval of integration. The accuracy of the result depends on the number of sample points considered; generally, the more the sample points, the greater the accuracy. There are two reasons why you might want to limit the accuracy of the integral:

- The length of time to calculate the integral increases as the number of sample points increases.
- There are inherent inaccuracies in each calculated value of $f(x)$:

- Experimentally derived constants in $f(x)$ may be inaccurate. For example, if $f(x)$ contains experimentally derived constants that are accurate to only two decimal places, it is of little value to calculate the integral to the full (12-digit) precision of the calculator.
- If $f(x)$ models a physical system, there may be inaccuracies in the model.
- The calculator itself introduces round-off error into each computation of $f(x)$.

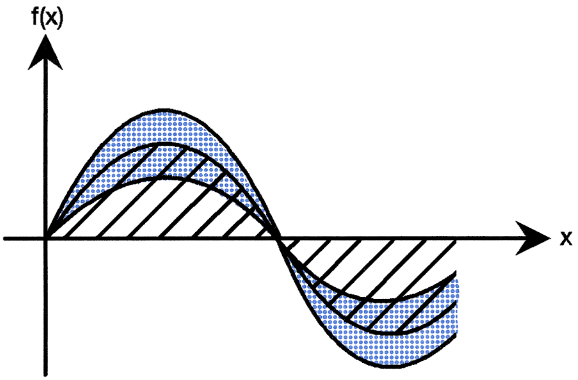
To indirectly limit the accuracy of the integral, you specify the *accuracy factor* of the *integrand* $f(x)$, defined as:

$$accuracy\ factor \leq \left| \frac{\text{true value of } f(x) - \text{computed value of } f(x)}{\text{computed value of } f(x)} \right|$$

The accuracy factor is your estimation in decimal form of the error in each computed value of $f(x)$. You specify the accuracy factor by setting the Display mode to n Fix. For example, if you set the display mode to 2 Fix, the accuracy factor is 0.01, or 1%. If you set the display mode to 5 Fix, the accuracy factor is 0.00001, or .001%.

The accuracy factor is related to the *uncertainty of integration* (a measurement of the accuracy of the *integral*) by:

$$uncertainty\ of\ integration \leq accuracy\ factor \times \int |f(x)|dx$$

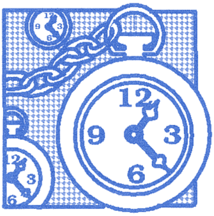


The striped area is the value of the integral. The dotted area is the value of the uncertainty of integration. You can see that at any point x, the uncertainty of integration is proportional to $f(x)$.

The numeric integration algorithm uses an iterative method, doubling the number of sample points in each successive iteration. When the algorithm stops, the current value of the integral is returned to level 1, and the uncertainty of integration is stored in the variable *IERR*. The error in the final value will almost certainly be less than the uncertainty of integration.

24

Time, Alarms, and Date Arithmetic



The Time application gives you a system clock that shows the current date and time. You can set alarms that either display messages or perform other actions you specify. You can also make time and date calculations.

24

Using the Clock (Date and Time)

When you display the clock, it appears in the upper-right corner of the display. It shows the current date and time in your choice of formats, shown in the table below. The formats also determine the way you enter dates and times in the command line. The following table illustrates how the clock shows 4:31 PM on February 21, 1992.

Display	Format	Command-Line
Date:		
02/21/1992	Month/day/year format	2.211992
21.02.1992	Day.month.year format	21.021992
Time:		
04:31:04P	12-hour format	4.3104
16:31:04	24-hour format	16.3104

Displaying the Date and Time

To display the date and time:

- To display them temporarily, press ⬅ TIME.
- To display them permanently, press ⬅ MODES NXT CLK. (To remove the permanent display, press CLK again.)

The date and time are always displayed while the TIME menu is active.

To change the date or time format:

24

1. Press ⬅ TIME SET.
2. Set the format:
   - To change the date format between month/day/year and day.month.year, press M/D.
   - To change the time format between 12-hour (AM and PM) and 24-hour, press 12/24.

Setting the Date and Time

To set the date:

1. Press ⬅ TIME SET.
2. Enter the date number in the command line using the current date format (month/day/year or day.month.year)—see the previous table.
3. Press →DAT.

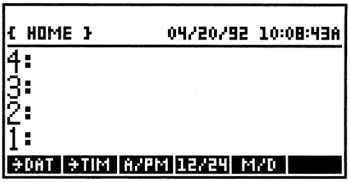
To set the time:

1. Press ⬅ TIME SET.
2. Enter the time number in the command line using the current time format (12-hour or 24-hour)—see the previous table.
3. Press →TIM.
4. To change the time between AM and PM, press A/PM.

Example: Set the date and time to 10:08 AM, April 20, 1992—or to the current date and time, if you want.

Get the TIME SET menu. (Make sure 12-hour and month/day/year formats are active.) Then set the date and time.

↱ CLR
↰ TIME SET
12/24 (if necessary)
M/D (if necessary)
10.08 →TIM
4.201992 →DAT



Note the new date and time in the status area.

24

To adjust the time:

1. Press ↰ TIME ADJST.
2. Enter the number of hours, minutes, or seconds to add or subtract from the current time (a positive number).
3. Press the "+" or "−" menu key for the unit and direction of the adjustment.

Example: To change standard time to daylight-saving time (1 hour later), press ↰ TIME ADJUST 1 HR+ .

Summary of Date and Time Operations

The TIME Menu—Clock Operations

Key	Programmable Command	Description
⬅ TIME SET:		
→DAT	→DATE	Sets the number in level 1 as the current date. The allowable range is January 1, 1989 to December 31, 2088.
→TIM	→TIME	Sets the number in level 1 as the current time. For 12-hour format, you can enter a 24-hour time number.
A/PM		Switches the clock setting between AM and PM. Adjusts the time for 24-hour format.
12/24		Switches between 12-hour and 24-hour format.
M/D		Switches between month/day/year and day.month.year format.
⬅ TIME ADJST:		
HR+		Increments the time by one hour.
HR-		Decrements the time by one hour.
MIN+		Increments the time by one minute.
MIN-		Decrements the time by one minute.
SEC+		Increments the time by one second.
SEC-		Decrements the time by one second.
CLKA	CLKADJ	Adds the specified number of *clock ticks* (positive or negative) to the time, where 8192 clock ticks equals 1 second. Use CLKADJ to change the calculator clock in a program.

Setting Alarms

You can set two types of alarms, which perform different actions when they come due:

- **Appointment alarm.** It displays the message you specified when you set the alarm. It also sounds a sequence of beeps for about 15 seconds—or until you press a key. You're expected to acknowledge an appointment alarm after it comes due.
- **Control alarm.** It executes the program or other object you specified when you set the alarm—no other action occurs. You don't acknowledge a control alarm.

24

When you set an alarm, it's saved in the system alarm list. You can review and edit alarms using the Alarm Catalog.

To see the next alarm due:

- Press (⬅)(TIME).

  or
- Press (⬅)(REVIEW) in the TIME menu.

Using Appointment Alarms

You can set an appointment alarm to display a message. If an event repeats periodically, you can specify a repeat interval.

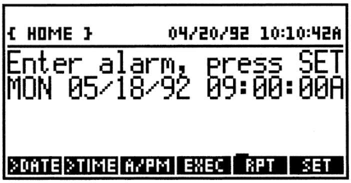
To set an appointment alarm:

1. Press (⬅)(TIME) ALRM.
2. If the alarm isn't for today, key in the alarm date using the current date format, then press >DATE.
3. Key in the alarm time using the current time format, then press >TIME.
4. To change the alarm time between AM and PM, press A/PM.
5. Optional: Key in a message (with " " delimiters), then press EXEC.
6. Optional: To make the alarm repeat at certain intervals, press RPT, key in the number of weeks, days, hours, minutes, or seconds, and press the menu key for the time unit—or press NONE to not repeat.
7. Press SET to set the alarm (and show the next due alarm).

Example: Appointment Alarm. Set an alarm for 9:00 AM on May 18, 1992, the time your report is due.

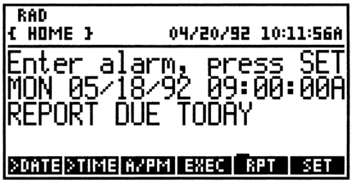
Set the alarm date and time.

⬅ TIME ALRM
5.181992 >DATE
9 >TIME



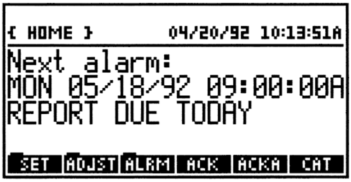
Enter an alarm message.

➡ " " REPORT DUE TODAY
EXEC



Set the alarm. The next due alarm is displayed.

SET



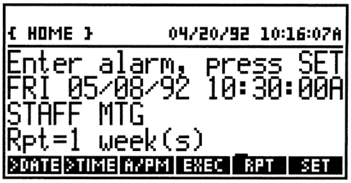
Example: Repeating Appointment Alarm. Set a repeating alarm for a weekly staff meeting on Fridays at 10:30 AM, beginning May 8, 1992.

Set the alarm time, date, and message.

⬅ TIME ALRM
5.081992 >DATE
10.30 >TIME
➡ " " STAFF MTG
EXEC

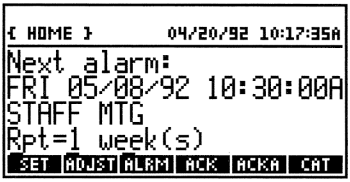
Set the repeat interval to 1 week.

RPT
1 WEEK



Set the alarm. The next due alarm is displayed.

SET



To respond to an appointment alarm:

- During the beeps, press any key, such as [ATTN].

  or
- After the beeps stop, press [↰][TIME] to see the message, then press ACK. (You can then press [ATTN] to return to the stack.)

When an appointment alarm comes due, the ((•)) annunciator turns on, the beeper sounds at short intervals for about 15 seconds, and the alarm message is displayed. If you press a key during the beeps, the alarm is acknowledged and deleted.

If you don't acknowledge an alarm during the beeps, the beeper stops and the message is cleared from the display. A repeating alarm is normally deleted automatically and rescheduled. A nonrepeating alarm becomes "past due," but not deleted—the ((•)) annunciator remains on to show you have a past-due alarm to respond to.

If you have several past-due alarms, you see the oldest one when you press [↰][TIME]. Each time you press ACK, the next oldest one is displayed. The ((•)) annunciator turns off when no past-due alarms remain.

To acknowledge all past-due alarms at once:

- Press ACKA in the TIME menu.

To stop a repeating alarm:

- See "Stopping Repeating Alarms" on page 24-9.

To save or not save nonrepeating alarms you acknowledge:

- To delete alarms when they're acknowledged, press 44 [+/-] [↱] [MODES] [NXT] CF .
- To save alarms when they're acknowledged, press 44 [+/-] [↱] [MODES] [NXT] SF .

Normally, a nonrepeating appointment alarm is deleted when you acknowledge it. Set system flag −44 to *save* it instead. However, having more than 20 past-due alarms may affect calculator performance—so it's a good idea to manage the number of alarms in the system alarm list. (Past-due repeating alarms are never saved.)

24

To change the way repeating alarms work:

- To automatically delete and reschedule them, press 43 [+/-] [↱] [MODES] [NXT] CF .
- To make them past-due and not reschedule them, press 43 [+/-] [↱] [MODES] [NXT] SF .

Normally, a repeating appointment alarm is automatically deleted and rescheduled. Set system flag −43 to change that behavior—the repeating alarm becomes past-due, and it isn't rescheduled until after you acknowledge it.

To control the alarm beeper:

- To enable the alarm beeper, press 57 [+/-] [↱] [MODES] [NXT] CF .
- To suppress the alarm beeper, press 57 [+/-] [↱] [MODES] [NXT] SF .

Normally, the alarm beeper sounds when an appointment comes due. Set system flag −57 to keep the beeper from sounding.

Using Control Alarms

You can set a control alarm to execute a program or other object. If you want to execute the object periodically, you can make the alarm repeat at a specified interval.

To set a control alarm:

1. Press (⬅)(TIME) ALRM .
2. If the alarm isn't for today, key in the alarm date using the current date format, then press >DATE.
3. Key in the alarm time using the current time format, then press >TIME.
4. To change the alarm time between AM and PM, press A/PM .
5. Enter the object or name to be executed, then press EXEC .
6. Optional: To make the alarm repeat at certain intervals, press RPT , key in the number of weeks, days, hours, minutes, or seconds, and press the menu key for the time unit—or press NONE to not repeat.
7. Press SET to set the control alarm.

24

You don't acknowledge a control alarm when it comes due—it's automatically considered to be acknowledged. Any control alarm (nonrepeating or repeating) that comes due is always *saved* in the system alarm list. Flags −43 and −44, which affect appointment alarms, have no effect on control alarms.

When a control alarm comes due, a copy of the *alarm index* is returned to level 1, then the specified object is executed. The alarm index is a real number that identifies the alarm based on its chronological order in the system alarm list—you can use it with programmable alarm commands, as described under "Using Alarms in Programs" on page 24-15.

To stop a repeating alarm, see the next topic.

Stopping Repeating Alarms

To delete a repeating alarm:

1. Press (⬅)(TIME) CAT to get the Alarm Catalog.
2. Press (▲) and (▼) as required to move the ▶ pointer to the alarm you want to delete.

3. Press PURG.
4. Press (ATTN).

The Alarm Catalog is described under "Reviewing and Editing Alarms" on page 24-12.

To recover from a short-interval repeating alarm:

- Press the (ON) and (4) keys simultaneously, then release them.

It's possible for a repeating alarm to have a short enough repeat interval that it reschedules and executes faster than you can delete it from the alarm list. This may occur if you mistakenly set a repeating
24 appointment alarm for a very short interval. It may also occur in the case of a control alarm that executes a program to take measurements at short intervals.

The (ON)-(4) keystroke sets a state in the calculator that cancels the rescheduling of the *next* due alarm (presumably the short-interval repeat alarm). When that alarm comes due—or when you press the next key—the special "no-reschedule" state of the calculator is canceled so future alarms aren't affected. Because pressing a key cancels the "no-reschedule" state, you should wait until the alarm comes due before pressing any keys.

To restart the short-interval repeat alarm at a future time, use the Alarm Catalog to edit the repeating alarm and set a new starting time. See the next topic for details.

Summary of Alarm Operations

The TIME Menus—Alarm Operations

Key	Programmable Command	Description
⬑ TIME:		
ALRM		Selects the ALRM menu for entering an alarm. The ALRM menu also contains commands for using alarms in programs.
ACK	ACK	Acknowledges the oldest past-due alarm.
ACKA	ACKALL	Acknowledges all past-due alarms.
CAT		Selects the Alarm Catalog for reviewing and editing existing alarms.
⬑ TIME ALRM (page 1):		
>DATE		Sets the number in level 1 as the alarm date. If the year digits are zero, the current year is used.
>TIME		Sets the number in level 1 as the alarm time.
A/PM		Switches the alarm time between AM and PM.
EXEC		Stores the object in level 1 as the alarm execution action. If the object is a string, the alarm is treated as an appointment alarm, displaying the contents of the string as the alarm message. If the object is *not* a string, the alarm is a control alarm, and the object is executed when the alarm comes due. (⮫ EXEC recalls the current object to the stack.)
RPT		Selects the RPT menu for setting a repeat interval.

24

The TIME Menus—Alarm Operations (continued)

Key	Programmable Command	Description
SET		Sets the alarm currently being constructed, and saves it in the system alarm list.
⬅ TIME ALRM RPT :		
WEEK DAY HOUR MIN SEC		Sets the repeat interval to the number of weeks, days, hours, minutes, or seconds specified in level 1.
NONE		Cancels the repeat interval.

24

Reviewing and Editing Alarms

You can review, edit, and delete future and past-due alarms in the Alarm Catalog. The Alarm Catalog is a special environment where the keyboard is redefined and limited to special operations.

To get the Alarm Catalog:

- Press CAT in the TIME menu.

or

- Press ➡ TIME (*right*-shift).

The Alarm Catalog displays the system alarm list with the ▶ pointer at the next alarm due. (If there are no alarms in the list, the message Empty catalog appears.)

To work with an alarm in the Alarm Catalog:

1. Press ▲ and ▼ as required to move the pointer to the alarm you want to use.
2. Perform the operation:
   - To delete the alarm, press PURG.
   - To view the alarm information, press VIEW.
   - To change the alarm information, press EDIT, then update and set the alarm as described under "Setting Alarms" on pages 24-5 and 24-9.

When you press EDIT, the selected alarm is *removed* from the alarm list—it's not returned there until you press SET. It is, however, saved in a reserved variable *ALRMDAT* until you press SET.

To exit the Alarm Catalog:

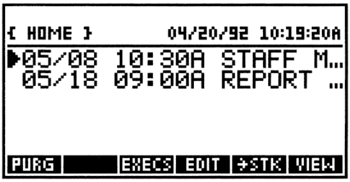
- Press (ATTN).

Example: Change the alarm for the staff meeting in the previous example from 10:30 AM to 9:30 AM on the same day.

Select the alarm catalog.

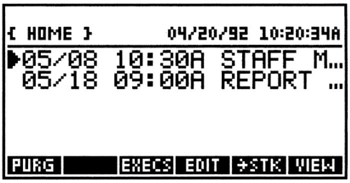
24

(↰)(TIME) CAT



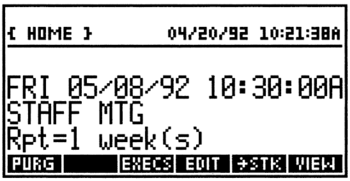
Move the pointer to the 10:30 alarm. (The position of the alarm in the catalog may vary depending on the specific dates you've used in previous examples.)

(▲) or (▼) (as required)



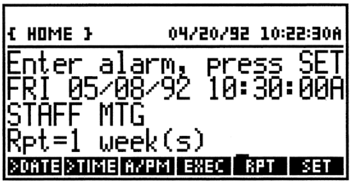
View the alarm to check if this is the one you want to edit.

VIEW



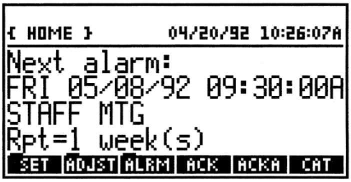
Start editing the alarm.

EDIT



Set the new alarm time. The next due alarm is displayed.

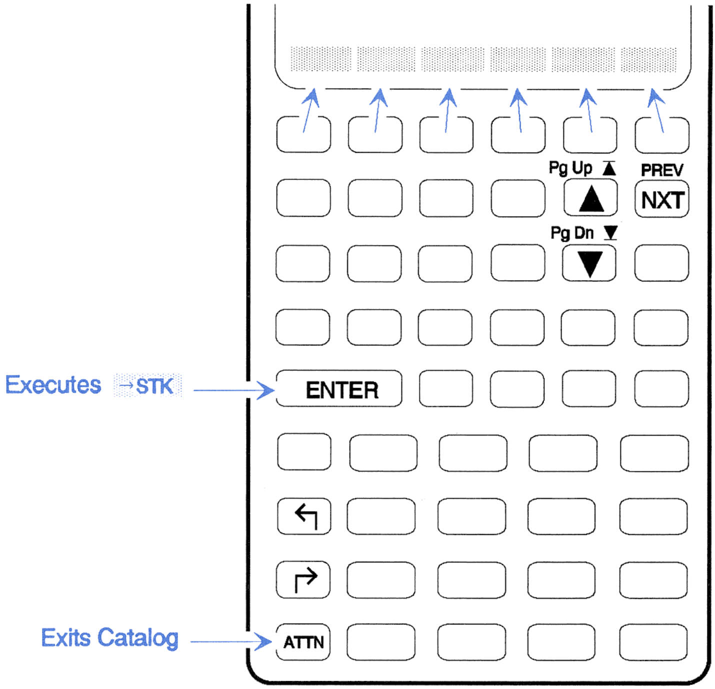
9.30 >TIME SET



The following table and illustration summarize the operations available in the Alarm Catalog.

Operations in the Alarm Catalog

Key	Description
PURG	Deletes the selected alarm from the alarm list.
EXECS	Switches between displaying the date and time of each alarm entry and displaying the alarm execution object only.
EDIT	Deletes the selected alarm from the system alarm list for editing and exits the catalog.
→STK	Copies the selected alarm to the stack.
VIEW	Views all information about the selected alarm.
▲ ▼	Moves the catalog pointer up or down one level. When prefixed with ◀, moves the catalog pointer up or down one page (◀ PgUp and ◀ PgDn in the following keyboard illustration). When prefixed with ▶, moves the catalog pointer to the top or bottom of the catalog (▶ ▲ and ▶ ▼ in the following keyboard illustration).
ENTER	Copies the selected entry to the stack (same as →STK).
ATTN	Exits the catalog.



24

Using Alarms in Programs

Many of the operations in the Time application that you execute from the keyboard aren't programmable. However, the application also includes several programmable commands that let you control alarms in programs.

You use a list to specify an alarm—it has the following form:

{ *date time action repeat* }

where *date* and *time* are the alarm date and time in the current formats, *action* is the execution object, and *repeat* is the repeat interval in *clock ticks* (1 clock tick is 1/8192 second).

Programmable Alarm Commands

Key	Programmable Command	Description
⬅ TIME ALRM (page 2):		
STOAL	STOALARM	Stores the alarm in level 1 into the system alarm list and returns its alarm index *n* to level 1. The argument for STOAL can take any one of the following four forms: ■ *time* (alarm date is current date) ■ { *date time* } ■ { *date time action* } ■ { *date time action repeat* }
RCLAL	RCLALARM	Takes an alarm index *n* from level 1, and returns the corresponding alarm to level 1.
DELAL	DELALARM	Takes an alarm index *n* from level 1 and deletes the corresponding alarm from the system alarm list. If $n = 0$, deletes *all* alarms from the system alarm list.
FINDA	FINDALARM	Returns the alarm index *n* of the first alarm that comes due after the time specified in level 1 as follows: if the level 1 argument is a list of the form { *date time* }, returns the first alarm due after that date and time; if the level 1 argument is a real number *date*, returns the first alarm due after midnight on that date; if the level 1 argument is 0, returns the first past-due alarm.

Calculating with Dates and Times

You can use the TIME menu to calculate calendar and clock intervals.

Making Date Calculations

To make a date calculation:

1. Press ⬅ TIME NXT.
2. Enter the arguments for the command:
   - Enter a date in command-line form using the current date format (*MM.DDYYYY* or *DD.MMYYYY*).
   - Enter a time in command-line form (24-hour *HH.MMSSs*).
   - Enter an interval as a real number of days (positive or negative).
3. Press the menu key for the command—see the table below.

24

The TIME Menu—Date Arithmetic Commands

Keys	Programmable Command	Description
⬅ TIME (page 2):		
DATE+	DATE+	Returns a past or future date in number form (*MM.DDYYYY* or *DD.MMYYYY*), given a date in level 2 and the number of days in level 1.
DDAYS	DDAYS	(Delta days.) Returns the number of days between the dates in level 2 and level 1.
DATE	DATE	Returns the current date in number form (*MM.DDYYYY* or *DD.MMYYYY*).
TSTR	TSTR	(Time string.) Returns a string object (characters) describing any valid date in level 2 and 24-hour time in level 1.

Example: Find the expiration date for a 120-day option purchased on July 15, 1991.

Get the TIME menu, enter the known date, key in the number of days, then calculate the expiration date.

[⬅] [TIME] [NXT]
7.151991 [ENTER]
120 DATE+

Example: Find the number of days between April 20, 1982 and August 2, 1986.

Get the TIME menu, enter the first and second dates, and calculate the number of days.

[⬅] [TIME] [NXT]
4.201982 [ENTER]
8.021986 DDAYS

Example: Find the date 90 days from today. (This example assumes the current date is April 20, 1992.)

Get the current date in level 1.

[⬅] [TIME] [NXT] DATE

Enter the number of days and calculate the future date. The result is July 19, 1992.

30
DATE+

Making Time and Angle Calculations

To make a time calculation:

1. Press [⬅] [TIME] [NXT] (as required).
2. Enter the time arguments for the command in HMS or decimal format, as required.
3. Press the menu key for the command—see the table below.

A number with HMS (hours-minutes-seconds) format is represented as *H.MMSSs*:

H	Zero or more digits representing the number of hours.
MM	Two digits representing the number of minutes.
SS	Two digits representing the number of seconds.
s	Zero or more digits representing the decimal fraction part of seconds.

To make an angle calculation:

1. Press ⬅ TIME NXT (as required).
2. Enter the angle arguments for the command in HMS or decimal format, as required.
3. Press the menu key for the command—see the table below.

24

For angle calculations, you can use angles in degrees-minutes-seconds (HMS) format—*H* in HMS format represents *degrees*. (See also "Angle Conversion Functions" on page 9-11.)

The TIME Menu—Time Arithmetic Commands

Key	Programmable Command	Description
⬅ TIME (pages 2 and 3):		
TIME	TIME	Returns the current time in 24-hour number form.
TICKS	TICKS	Returns the system time as a binary integer in units of 1/8192 second.
→HMS	→HMS	Converts a real number representing decimal hours (or degrees) to HMS format.
HMS→	HMS→	Converts a real number representing hours (or degrees) in HMS format to its decimal form.
HMS+	HMS+	Adds two numbers in HMS format, returning the sum in HMS format.
HMS−	HMS−	Subtracts two numbers in HMS format, returning the difference in HMS format.

Example: Convert 5.27 hours to its HMS equivalent.

Key in the decimal time and execute the conversion. The answer is 5 hours, 16 minutes, 12 seconds.

(←) (TIME) (NXT) (NXT)
5.27 →HMS

Example: Add 5°50′ (5 degrees 50 minutes) and 4°30′.

Enter the two angles in HMS format and add them. The answer is 10°20′.

(←) (TIME) (NXT) (NXT)
24 5.5 (ENTER)
4.3 HMS+

To calculate elapsed time in a program:

- Enter a TICKS command at the start and end of the program segment you want to time, then subtract the two times. If desired, multiply by 8192 to get the number of seconds.

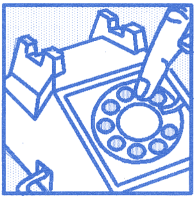
Example: See the *FIBT* program on page 31-5.

Part 4

Programming

25

Programming Fundamentals



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

25

Understanding Programming

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

For example, if you want to find the negative square root of a number that's in level 1, you might press [√x] [+/-]. The following program executes the same commands:

```
« √ NEG »
```

Without changing the program, we *could* show it with one command per line—similar to other programming languages:

```
«
 √
 NEG
»
```

The next few topics introduce certain aspects of programs:

- Contents.
- Calculations.
- Structured programming.

Each of these ideas is explained in detail in chapters 25 through 31—these next few topics give just an overview.

The Contents of a Program

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

Actions for Certain Objects in Programs

Object	Action
Command	*Executed.*
Number	Put on the stack.
Algebraic	Put on the stack.
String	Put on the stack.
List	Put on the stack.
Program	Put on the stack.
Global name (quoted)	Put on the stack.
Global name (unquoted)	■ Program *executed.* ■ Name evaluated. ■ Directory becomes current. ■ Other object put on the stack.
Local name (quoted)	Put on the stack.
Local name (unquoted)	Contents put on the stack.

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

Examples of Program Actions

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

Actually, programs can contain more than just objects—they can also contain *structures*. A structure is a program segment with a defined organization. Two basic kinds of structures are available: 25

- **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—sometimes put there by the user or by another program. Here are two typical ways to manipulate that data:

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

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


```
« 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.

Local variable structures with algebraic objects are favored by many programmers because they're easy to write, easy to read, and simple to debug.

Structured Programming

The HP 48 encourages *structured programming*. Every program has only one entrance point—the beginning of the program. It also has only one exit point—the end of the program. There are no labels inside a program to jump to—there are no GOTO commands to exit from. From an external point of view, program flow is extremely simple—start at the beginning, stop at the end. (Of course, *inside* the program you can use branching structures to control the execution flow.)

You can take advantage of structured programming by creating "building-block" programs. Each building-block program can stand alone—and it can act like a subroutine in a larger program. For example, consider the following program:

```
« GETVALUE FINDANSWER OUTANSWER »
```

This program is separated into three main tasks, each with a subroutine. The flow is predictable. Only the input and output of each subroutine matter—the internal workings don't matter at this level.

Within each subroutine, its task can be simple—or it can be subdivided further into other subroutines that perform smaller tasks. This lets you have relatively simple subroutines—even if your main program is large.

So, programs become extensions to the set of built-in commands, as mentioned earlier. You execute them by name. They take certain inputs, and they produce certain results.

Where to Find More Information

To find information about certain programming topics, look in the following chapters:

- Entering, editing, and running programs—this chapter.
- Creating basic programs—this chapter.
- Testing programs—this chapter.
- Using tests and conditional structures—chapter 26.
- Using loop structures—chapter 27.
- Using flags—chapter 28.
- Creating interactive programs—chapter 29.
- Trapping errors in programs—chapter 30.
- Checking out example programs—chapter 31.
- Finding information about commands—appendix G.

You can also refer to these books:

- The *Programmer's Reference Manual* for the HP 48 (part number 00048-90054) contains programming information, including syntax information for all HP 48 commands, in a reference format.

- *HP 48 Programming Examples* by D.R. Mackenroth, Addison-Wesley, 1991, is a source of structured programs and programming techniques.

Entering and Executing Programs

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

To enter a program:

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

In Program-entry mode (PRG annunciator on), command keys aren't executed—they're entered in the command line instead. Only nonprogrammable operations such as [◆] and [VAR] are executed.

Line breaks are discarded when you press [ENTER].

To enter commands and other objects in a program:

- Press the keyboard or menu key for the command or object.

  or
- Type the characters using the alpha keyboard.

To store or name a program:

1. Enter the program on the stack.
2. Enter the variable name (with ' delimiters) and press [STO].

You can choose descriptive names for programs. Here are some ideas of what the name can describe:

- The calculation or action. Examples: *SPH* (spherical-cap volume), *SORT* (sort a list).
- The input and output. Examples: *X→FX* (x to $f(x)$), *RH→V* (radius-and-height to volume).
- The technique. Examples: *SPHLV* (spherical-cap volume using local variables), *SPHSTACK* (spherical-cap volume using the stack).

To execute a program:

- Press [VAR] then the menu key for the program name.

 or
- Enter the program name (with *no* delimiters) and press [ENTER].

 or
- Put the program name in level 1 and press [EVAL].

 or
- Put the program object in level 1 and press [EVAL].

To stop an executing program:

- Press [ATTN].

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 [×] 3 [÷] [▶][→NUM]

Enter the same keystrokes in a program. ([▶][↵] just starts a new line.)

[◀][« »]
3 [y^x] [◀][π] [×] 4 [×] 3 [÷]
[▶][↵] [▶][→NUM]

```
« 3 ^ π * 4 * 3 /
→NUM ◆
»
PARTS PROB HYP MATR VECTR BASE
```

Put the program on the stack.

[ENTER]

```
1: « 3 ^ π * 4 * 3 /
   →NUM »
←SKIP SKIP→ ←DEL DEL→ INS ■ ↑STK
```

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

```
1:          268.082573107
VOL
```

The program is

```
« 3 ^ π * 4 * 3 / →NUM »
```

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


```
« → r '4/3*π*r^3' →NUM »
```

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

Enter the program. ([↱][↵] just starts a new line.)

[↰][« »]
[↱][→] r [SPC]
['] 4 [÷] 3 [×] [↰][π] [×]
r [y^x] 3 [▶] [↱][↵]
[↱][→NUM]

```
« → r '4/3*π*r^3'
→NUM ◆
»
VOL
```

Put the program on the stack and store it in *VOL*.

[ENTER]
[↰] VOL

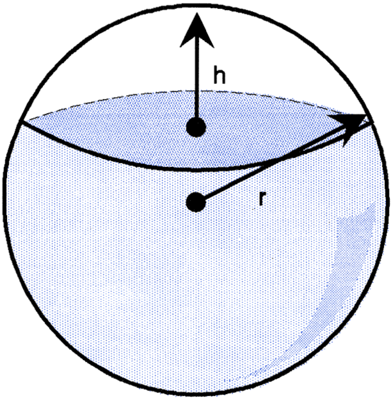
```
VOL
```

Calculate the volume for a radius of 4.

4 VOL

```
1:          268.082573106
VOL
```

Example: Enter a program *SPH* that calculates the volume of a spherical cap of radius r and height h using values stored in variables R and H.



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

In this and following chapters on programming, "stack diagrams" show what arguments must be on the stack before a program is executed and what results the program leaves on the stack. Here's the stack diagram for *SPH*. **25**

Arguments	Results
	1: *volume*

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

Program listings are shown with program steps in the left column and associated comments in the right column. Remember, you can either press the command keys or type in the command names to key in the program. In this first listing, the keystrokes are also shown.

Program:	Keys:	Comments:
«	[←] [« »]	Begins the program.
'1/3	['] 1 [÷] 3	Begins the algebraic expression to calculate the volume.
*π*H^2	[×] [←] [π] [×] H [y^x] 2	Multiplies by πh^2.
*(3*R-H)'	[×] [←] [( )] 3 [×] R [−] H [▶] [▶]	Multiplies by $3r - h$, completing the calculation and ending the expression.
→NUM	[→] [→NUM]	Converts the expression with π to a number.
»		Ends the program.
	[ENTER]	Puts the program on the stack.
	['] SPH [STO]	Stores the program in variable *SPH*.

25

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] SPH

```
1:          254.469004942
 H  | R  | SPH | VOL |   |
```

Viewing and Editing Programs

You view and edit programs the same way you view and edit other objects—using the command line. See "Displaying Objects For Viewing and Editing" on page 3-6.

To view or edit a program:

1. View the program:
   - If the program is in level 1, press ⬅ EDIT.
   - If the program is stored in a variable, put the variable name in level 1 and press ➡ VISIT.
2. Optional: Make changes.
3. Press ENTER to save any changes (or press ATTN to discard changes) and return to the stack.

➡ VISIT lets you change a stored program without having to do a store operation. ⬅ EDIT lets you change a program and then store the new version in a different variable. 25

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

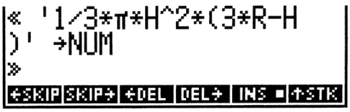
To switch between entry modes:

- Press ➡ ENTRY.

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

Use VISIT to start editing *SPH*.

VAR
' SPH
➡ VISIT



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

▶ ' H ▶ STO
' R ▶ STO

```
«'H' STO 'R' STO ◆1/3…
)' →NUM
»
←SKIP SKIP→ ←DEL DEL→ INS ■ ↑STK
```

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

ENTER
' SPH ↱ VISIT

```
« 'H' STO 'R' STO '
1/3*π*H^2*(3*R-H)'
→NUM
»
←SKIP SKIP→ ←DEL DEL→ INS ■ ↑STK
```

Press ATTN to stop viewing.

Creating Programs on a Computer

Some people find it convenient to create programs and other objects on a computer, then load them into the HP 48 using its serial port. For example, the Program Development Link from Hewlett-Packard provides a program-development environment tailored to the HP 48. You can use it to create programs, send them to the calculator, and run them.

Also, if you're creating programs on a computer, you can include "comments" in the computer version of the program.

To include a comment in a program:

- Enclose the comment text between two @ characters.

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

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

Using Local Variables

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

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

Local variables address the disadvantages of global variables in programs. Local variables are temporary variables *created by a program*. They exist only while the program is being executed and cannot be used outside the program. They never appear in the VAR menu. In addition, local variables are accessed faster than global variables. (By convention, this manual uses lowercase names for local variables.)

25

Creating Local Variables

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

To enter a local variable structure in a program:

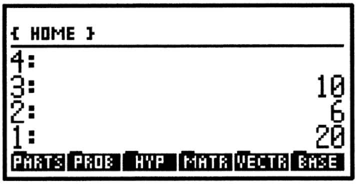
1. Enter the → command (press [→] [→]).
2. Enter one or more variable names.
3. Enter a *defining procedure* (an algebraic or program object) that uses the names.

« → $name_1$ $name_2$ ... $name_n$ '*algebraic*' »

or

« → $name_1$ $name_2$ ... $name_n$ « *program* » »

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



then

→ a creates local variable $a = 20$.

→ a b creates local variables $a = 6$ and $b = 20$.

→ a b c creates local variables $a = 10$, $b = 6$, and $c = 20$.

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

Local variable structures have these advantages:

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

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

Arguments	Results
2: *r* 1: *h*	1: *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	1: 254.469004942 SPHLV H R SPH VOL

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 in this procedure.
`a b + c +`	
`→ 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: In the following program excerpt, the defining procedure for local variables *d*, *e*, and *f* calls a program that you previously created and stored in global variable *P1*.

Program:	**Comments:**
«	
⋮	
→ a b c	
«	
a b + c +	
→ d e f	Defines local variables *d*, *e*, *f*.
'P1+a/(d*e+f)'	Local variables *a*, *b*, *c* and *d*, *e*, *f* are available in this procedure. The defining procedure executes the program stored in variable *P1*.
a c / -	
»	
⋮	
»	

25

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

Conversely, program *P1* can create its own local variable structure (with any names, such as *a*, *c*, and *f*, for example) without conflicting with the local variables of the same name in the procedure that calls *P1*.

Creating User-Defined Functions as Programs

The defining procedure for a local variable structure can be either an algebraic or program object. As discussed in "The Structure of a User-Defined Function" on page 10-5, a user-defined function is a program that consists solely of a local variable structure whose defining procedure is an algebraic expression.

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, the loop structures described in chapter 27 are not allowed in algebraic expressions.

Example: Program *BER* on page 31-34 calculates a Bessel function approximation. *BER* uses a local variable structure whose defining procedure is a program containing a FOR...STEP structure and a nested IF...THEN...ELSE...END structure. *BER* is not differentiable, but the example in chapter 31 demonstrates that it can take its arguments either from the stack or in algebraic syntax.

25

Manipulating Data on the Stack

The programs *SPH* and *SPHLV* earlier in this chapter use variables for data storage and recall. An alternative programming method manipulates numbers on the stack without storing them in variables. Although this method may give faster program execution in certain situations, there are certain disadvantages of the stack-manipulation method:

- As you write a program, you must keep track of the location of the data on the stack. For example, data arguments must be duplicated if they're used by more than one command.
- A program that manipulates data on the stack is generally harder to read and understand than a program that uses variables.

Example: The following program *SPHSTACK* uses the stack-manipulation method to calculate the volume of a spherical cap. (*SPHLV* uses local variables to execute the same calculation in about 30 percent less time.)

Arguments	Results
2: *r* 1: *h*	1: *volume*

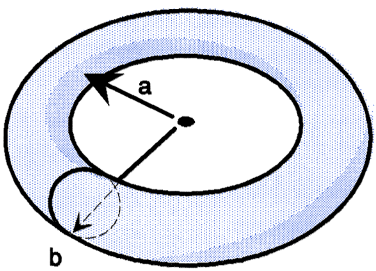
Program:	**Comments:**
`«`	
`SWAP`	Puts the radius in level 1.
`3 *`	Multiplies the radius by 3.
`OVER -`	Copies the number in level 2 (the height) to level 1 and subtracts, calculating $3r - h$.
`SWAP SQ *`	Swaps the original height into level 1, squares it, and multiplies by $3r - h$.
`π * 3 /`	Multiplies by π and divides by 3, completing the calculation.
`→NUM`	Converts algebraic to a number.
`»`	
ENTER	Puts the program on the stack.
' SPHSTACK STO	Stores it in *SPHSTACK*.

25

Using Subroutines

Because a program is itself an object, it can be used in another program as a subroutine. When program B is used by program A, program A *calls* program B, and program B is a *subroutine* in program A.

Example: The program *TORSA*, calculates the surface area of a torus of inner radius a and outer radius b. *TORSA* is used as a subroutine in a second program *TORSV*, which calculates the volume of a torus.



The surface area and volume are calculated by

$$A = \pi^2(b^2 - a^2) \qquad V = \frac{1}{4}\pi^2(b^2 - a^2)(b - a)$$

(The quantity $\pi^2(b^2 - a^2)$ in the second equation is the surface area of a torus calculated by *TORSA*.)

Here are the stack diagram and program listing for *TORSA*.

Arguments	Results
2: *a*	
1: *b*	1: *surface area*

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]	Puts the program on the stack.
['] TORSA [STO]	Stores the program in *TORSA*.

25

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

Arguments	Results
2: *a*	
1: *b*	1: *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]	Puts the program on the stack.
['] TORSV [STO]	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
TORSV TORSA SPHST SPHLV  H  R
```

Single-Stepping through a Program

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

To single-step from the start of a program:

1. Put the program or program name in level 1 (or the command line).
2. Press [PRG] CTRL DBUG to start and immediately suspend execution. The HALT annunciator is displayed in the status area.
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.

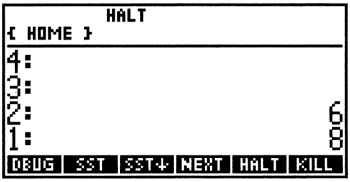
To turn off the HALT annunciator at any time:

- Press (PRG) CTRL KILL .

Example: Execute program *TORSV* step by step. Use $a = 6$ and $b = 8$.

Select the VAR menu and enter the data. Enter the program name and start the debugging. The HALT indicates program execution is suspended.

(➡)(CLR)
(VAR)
6 (ENTER) 8 (ENTER)
(') TORSV
(PRG) CTRL DBUG



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

SST

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

SST ... SST

To single-step from the middle of a program:

1. Insert a HALT command in the program where you want to begin single-stepping.

2. Execute the program normally. The program stops when the HALT command is executed, and the HALT annunciator is displayed.
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:

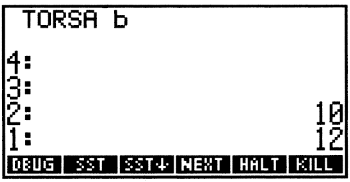
- To execute the subroutine in one step, press SST.
- To execute the subroutine step-by-step, press SST↓.

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.

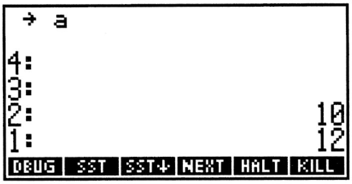
Select the VAR menu and enter the data. Enter the program name and start the debugging. Execute the first four steps of the program, then check the next step.

(↱)(CLR) (VAR)
10 (ENTER) 12
(') TORSV
(PRG) CTRL DBUG
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.

Single-Step Operations

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

26

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" on page 26-4.

Test commands separate into three groups:

- **Comparison functions.** Compare two objects.
- **Logical functions.** Combine the results of previous tests.
- **Flag-test commands.** Test the states of flags, as described in chapter 28, "Flags."

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):		
<	$<$	Less than.
>	$>$	Greater than.
≤	$\leq$	Less than or equal to.
≥	$\geq$	Greater than or equal to.
==	==	Tests equality of two objects.
≠	$\neq$	Not equal.
SAME	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.

26

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

All comparison commands except SAME return the following:

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

- If one object is an algebraic (or name) and the other object is an algebraic (or name) or a number, returns an expression that must be evaluated to get a test result based on numeric values. For example, if 6 is stored in X, `'X' 5 <` returns `'X<5'`, then →NUM returns 0.

(Note that == is used for comparisons, while = separates two sides of an equation.)

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

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

26

Using Logical Functions

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

Logical Functions

Keys	Programmable Command	Description
PRG TEST (page 1):		
AND	AND	Returns 1 (true) only if both arguments are true.
OR	OR	Returns 1 (true) if either or both arguments are true.
XOR	XOR	Returns 1 (true) if either argument, but not both, is true.
NOT	NOT	Returns 1 (true) if the argument is 0 (false); otherwise, returns 0 (false).

AND, OR, and XOR combine two test results. For example, if 4 is stored in Y, `Y 8 < 5 AND` returns 1. First, `Y 8 <` returns 1 to the stack. AND removes 1 and 5 from the stack, interpreting both as true results, and returns 1 to the stack.

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

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

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

Testing Object Types

The TYPE command ((PRG) TEST 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 on page 4-19 to find HP 48 objects and their corresponding type numbers.

26

Using Conditional Structures and Commands

Conditional structures let a program make a decision based on the result of one or more tests. Conditional structures are built with commands—called structure words—that work only when used in proper combination with each other.

Conditional commands let you execute a decision-making process in which the true-clause and false-clause are each 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.

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

- Press (PRG) BRCH (⬅) IF .

See "Conditional Examples" on page 26-7.

26

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.

To enter IFT in a program:

- Press (PRG) BRCH (⬅)(PREV) IFT .

See "Conditional Examples" on page 26-7.

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

The syntax for this structure is

`« … IF` *test-clause*
`THEN` *true-clause* `ELSE` *false-clause* `END' … »`

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

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

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

- Press (PRG) BRCH (→) IF .

See "Conditional Examples" on page 26-7.

The IFTE Function

The algebraic syntax for this function is

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

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

You can also use the IFTE function with stack syntax. It takes three arguments: a *test-result* in level 3, a *true-clause* object in level 2, and a *false-clause* object in level 1.

To enter IFTE in a program or in an algebraic:

- Press (PRG) BRCH (←)(PREV) IFTE .

See "Conditional Examples" on page 26-7.

The CASE...END Structure

The syntax for this structure is

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

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

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

To enter CASE...END in a program:

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

See "Conditional Examples" below.

26

Conditional Examples

These examples illustrate conditional structures in programs.

Example: One Conditional Action. Both 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.
`»`	
`»`	

26

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 » »
```

Example: Two Conditional Actions. This program takes a value x from the stack and calculates sin x/x. At $x = 0$ the division would error, so the program returns the limit value 1 in this case.

```
« → x « IF 'x≠0' THEN x SIN x / ELSE 1 END » »
```

The following version uses IFTE algebraic syntax:

```
« → x 'IFTE(x≠0,SIN(x)/x,1)' »
```

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

Program:	Comments:
`«`	
`→ n1 n2`	Creates the local variables.
`«`	Starts the defining procedure.
`IF`	Starts the test clause.
`'n1≠0 AND n2≠0'`	Tests *n1* and *n2*.
`THEN`	If both numbers are nonzero,
`n1 n2 *`	multiplies the two values.
`ELSE`	Otherwise, returns the string
`"ZERO"`	ZERO.
`END`	Ends the conditional.
`»`	Ends the defining procedure.
`»`	

Example: Two Conditional Actions. This program tests if two numbers on the stack have the same value. If so, it drops one of the numbers and stores the other in variable V1—otherwise, it stores the number from level 1 in V1 and the number from level 2 in V2.

Program:	Comments:
`«`	
`IF` `DUP2` `SAME`	For the test clause, copies the numbers in levels 1 and 2 and tests if they have the same value.
`THEN` `DROP` `'V1' STO`	For the true clause, drops one of the numbers and stores the other in *V1*.
`ELSE` `'V1' STO` `'V2' STO`	For the false clause, stores the level 1 number in *V1* and the level 2 number in *V2*.
`END`	Ends the conditional structure.
`»`	
[ENTER]	Puts the program on the stack.
['] TST [STO]	Stores it in *TST*.

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

26 [ENTER] 52 [VAR] TST	V2 | V1 | TST | TORSV | TORSA | SPHST

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.
`»`	

Example: Stack Syntax. This program takes a value from level 1 and displays `POSITIVE` if it is positive or zero, and `NEGATIVE` otherwise: (The $\geq$ command compares the number with 0 and returns a test result for the IFTE command.)

```
« 0 ≥ "POSITIVE" "NEGATIVE" IFTE »
```

Example: Algebraic Syntax. This program is a user-defined function that takes a number (x) from the stack and calculates $\sin(x)/x$. If x is 0, the program returns 1.

27

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:

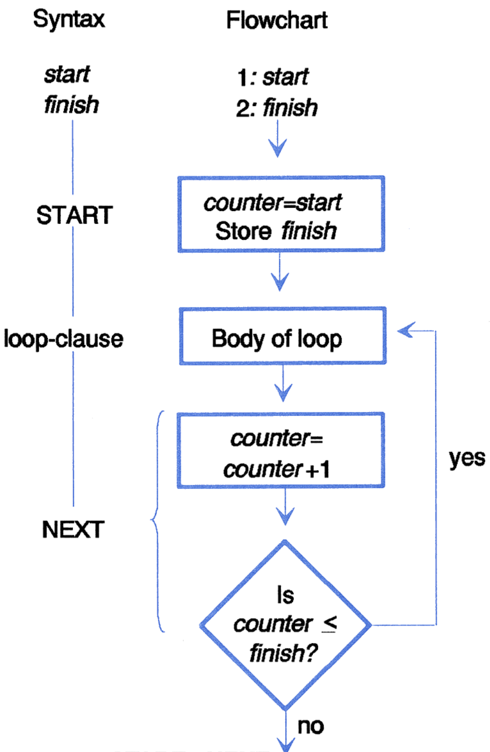
- NEXT. The counter increases by 1 for each loop.
- STEP. The counter increases or decreases by a specified amount for each loop.

The START...NEXT Structure

The syntax for this structure is

« ... *start finish* START *loop-clause* NEXT ... »

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



START...NEXT Structure

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

To enter START...NEXT in a program:

- Press (PRG) BRCH (←) START.

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

```
« 1 10 START "ABC" NEXT 10 →LIST »
```

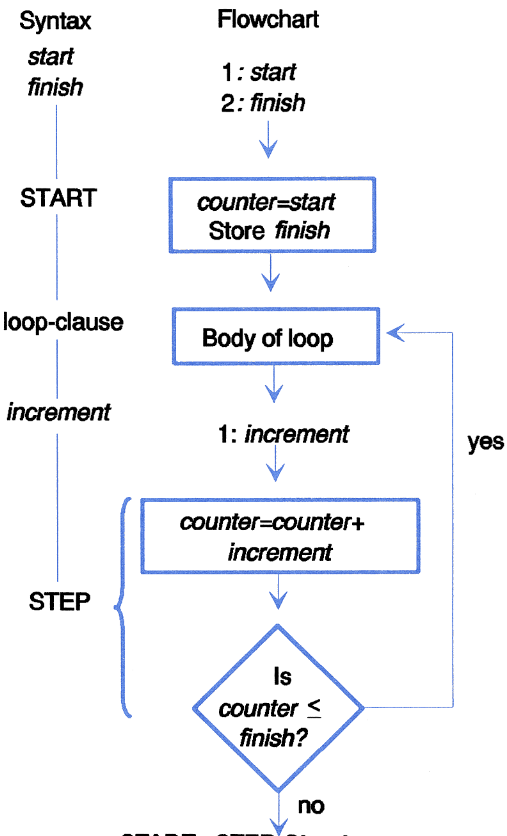
27

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 *final*. If the increment value is negative, the loop is executed if the counter is greater than or equal to *final*. Otherwise, execution resumes following STEP. In the following 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):

27

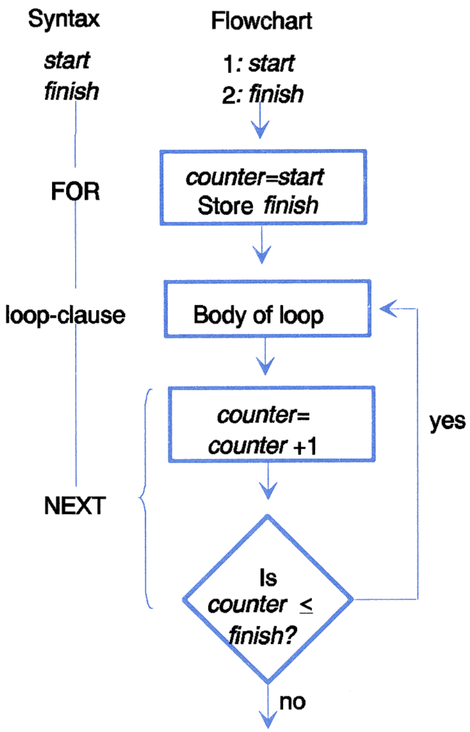
```
« DUP → x « x 1 START x SQ -3 STEP » »
```

The FOR...NEXT Structure

The syntax for this structure is

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

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



FOR...NEXT 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. Then the loop-clause is executed—*counter* can appear within the loop-clause. NEXT increments *counter-name* by one, and then tests whether its value is less than or equal to *finish*. If so, the *loop-clause* is repeated (with the new value of *counter*)—otherwise, execution resumes following NEXT. When the loop is exited, *counter* is purged.

To enter FOR...NEXT in a program:

- Press [PRG] BRCH [←] FOR .

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

```
« 1 5 FOR j j SQ NEXT »
```

Example: The following program takes the value x from the stack and computes the integer powers i of x. For example, when $x = 12$ and *start* and *finish* are 3 and 5 respectively, the program returns 12^3, 12^4, and 12^5. It requires as inputs *start* and *finish* in levels 3 and 2, and x in level 1. (→ x removes x from the stack, leaving *start* and *finish* there as arguments for FOR.)

```
« → x « FOR n 'x^n' EVAL NEXT » »
```

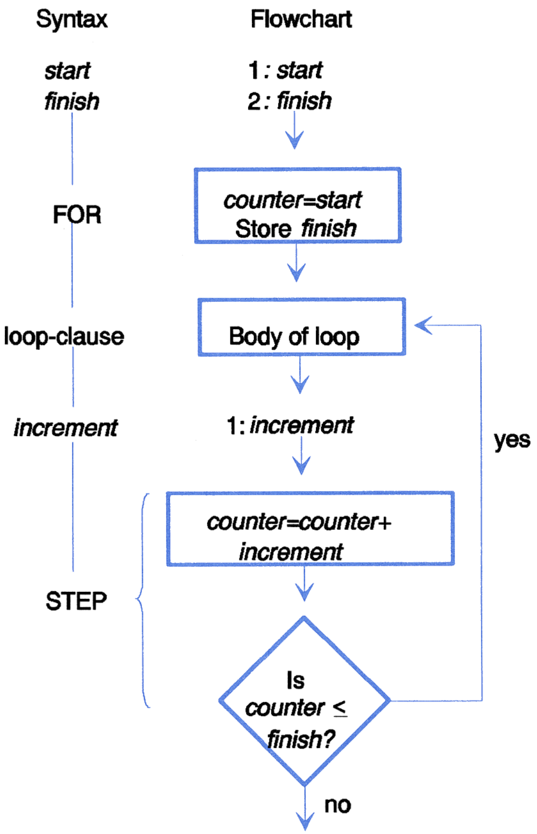
27

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 *final*. If the increment is negative, the loop is executed if *counter* is greater than or equal to *final*. Otherwise, *counter* is purged and execution resumes following STEP. In the following flowchart, the increment value is positive.

To enter FOR...STEP in a program:

- Press (PRG) BRCH (↱) FOR .

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

```
« 1 9 FOR x x SQ 2 STEP »
```

27

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

```
« 1 SWAP FOR n n n STEP »
```

Using Indefinite Loop Structures

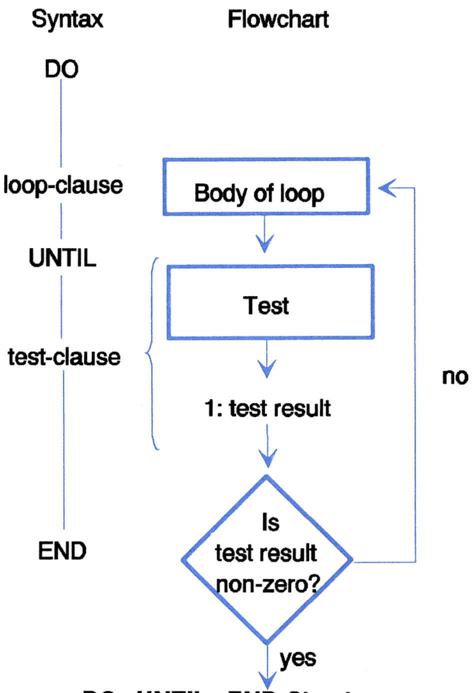
The DO...UNTIL...END Structure

The syntax for this structure is

« ... `DO` *loop-clause* `UNTIL` *test-clause* `END` ... »

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

27



DO...UNTIL...END Structure

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

To enter DO...UNTIL...END in a program:

- Press (PRG) BRCH (⬅) DO .

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

Program:	Comments:
`«`	
`DUP 1`	Duplicates n, stores the value into
`→ n s c`	n and s, and initializes c to 1.
`«`	Starts the defining procedure.
`DO`	Starts the loop-clause.
`'c' INCR`	Increments the counter by 1. (See "Using Loop Counters" on page 27-13.)
`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.
`»`	

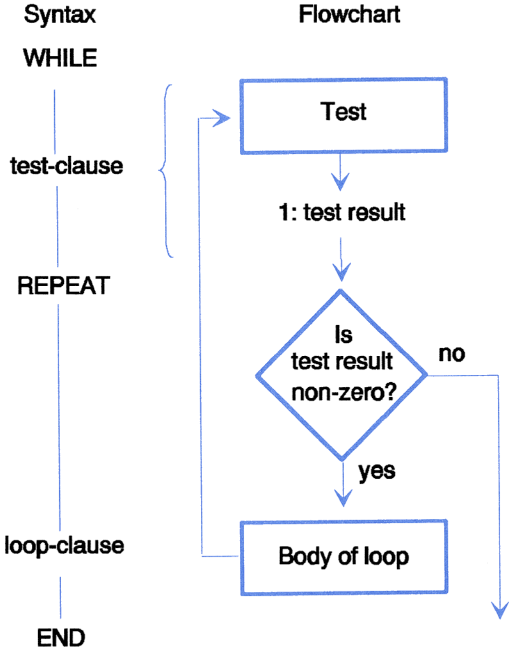
27

The WHILE...REPEAT...END Structure

The syntax for this structure is

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

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



WHILE...REPEAT...END Structure

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

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

- Press (PRG) BRCH (⬅) WHILE.

Example: The following program starts with a number on the stack, and repeatedly performs a division by 2 as long as the result is evenly divisible. For example, starting with the number 24, the program computes 12, then 6, then 3.

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

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


```
« WHILE DUP TYPE 3 == REPEAT Σ+ END »
```

Using Loop Counters

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

The syntax for INCR and DECR are

« ... '*variable*' INCR ... »
or
« ... '*variable*' DECR ... »

To enter INCR or DECR in a program:

- Press (↱)(MEMORY) `INCR` or `DECR`.

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

1. Changes the value stored in the variable by +1 or −1.
2. Returns the new value to the stack.

Example: If *c* contains the value 5, then `'c' INCR` stores 6 in *c* and returns 6 to the stack.

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

Program:	Comments:
`«`	
`0 → c`	Stores 0 in local variable *c*.
`«`	Starts the defining procedure.
`WHILE`	Starts the test clause.
`DUP TYPE 3 ==`	Returns true if level 1 contains a vector.
`'c' INCR`	Increments and returns the value in *c*.
`5 ≤`	Returns true if the counter $c \leq 5$.
`AND`	Returns true if the two previous test results are true.
`REPEAT`	
`Σ+`	Adds the vector to ΣDAT.
`END`	Ends the structure.
`»`	Ends the defining procedure.
`»`	

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

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

One program uses a FOR...NEXT loop—the other uses Σ.

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`	
`'Σ(j=1,n,j^2)'`	Uses Σ to calculate the summation.
`»`	

Example: The following program uses Σ 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.

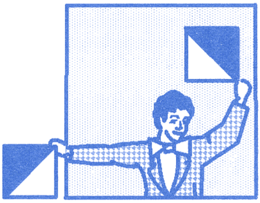
27

Program:	**Comments:**
«	
EVAL DUP SIZE OBJ→	Finds the dimensions of the array and the number of dimensions.
IF 1 ==	Tests for one dimension (vector).
THEN	For a vector, sums the vector elements.
→ a n	
'Σ(j=1,n,a(j))'	
ELSE	For a matrix, sums the matrix elements.
→ a m n	
'Σ(j=1,m,Σ(k=1,n, a(j,k)))'	
END	
»	

27

28

Flags



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

28

Types of Flags

The HP 48 has two types of flags:

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

Appendix E lists the 64 system flags and their definitions. For example, system flag −40 controls the clock display—when this flag is *clear* (the default state), the clock is displayed only when the TIME menu is selected—when this flag is *set*, the clock is displayed at all times. (Actually, when you press CLK in the MODES menu, you set or clear flag −40.)

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

Setting, Clearing, and Testing Flags

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

To set, clear, or test a flag:

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

Flag Commands

Key	Programmable Command	Description
PRG TEST (page 3) or ⮫ MODES (pages 2 and 3):		
SF	SF	Sets the flag.
CF	CF	Clears the flag.
FS?	FS?	Returns 1 (true) if the flag is set, or 0 (false) if the flag is clear.
FC?	FC?	Returns 1 (true) if the flag is clear, or 0 (false) if the flag is set.
FS?C	FS?C	Tests the flag (returns true if the flag is set), then clears the flag.
FC?C	FC?C	Tests the flag (returns true if the flag is clear), then clears the flag.

28

Example: System Flag. The following program sets an alarm for June 6, 1991 at 5:05 PM. It first tests the status of system flag −42 (Date Format flag) in a conditional structure and then supplies the alarm date in the current date format, based on the test result.

Program:	**Comments:**
«	
IF	Tests the status of flag −42, the Date Format flag.
-42 FC?	
THEN	If flag −42 is clear, supplies the date in *month/day/year* format.
6.151991	
ELSE	If flag −42 is set, supplies the date in *day.month.year* format.
15.061991	
END	Ends the conditional.
17.05 "TEST COMPLETE"	Sets the alarm: 17.05 is the alarm time and "TEST COMPLETE" is the alarm message.
3 →LIST STOALARM	
»	

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.

28

Program:	**Comments:**
«	
IF	Starts the conditional.
10 FS?	Tests the status of user flag 10.
THEN	If flag 10 is set, returns the integer part.
IP	
ELSE	If flag 10 is clear, returns the fractional part.
FP	
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 one or more flags during execution, you may also want it to save and restore the original flag states.

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

To recall the current flag states:

- Execute RCLF (⮫ [MODES] [NXT] RCLF).

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

{ #n_s #n_u }

To change the current flag states:

1. Enter the flag-state argument—see below.
2. Execute STOF (⮫ [MODES] [NXT] STOF).

STOF sets the current states of flags based on the flag-state argument:

#n_s	Changes the states of only the system flags.
{ #n_s #n_u }	Changes the states of the system and user flags.

Example: The program *PRESERVE* on page 31-8 uses RCLF and STOF.

29

Interactive Programs

Simple programs like those in chapter 25 use data supplied *before* program execution and return results as simple numbers. Such programs may be difficult to use, particularly if they're not documented. You must know what arguments to enter and in what order, and you must know how to interpret the results returned to the stack.

If you use *interactive* programs, they can prompt for data, display results with explanatory messages or tags, and allow you to choose how to proceed.

29

Stopping for Data Input

A program can stop for user input, then resume execution. You can use several commands to prepare for and suspend execution:

- PROMPT (⬅ CONT to resume).
- DISP FREEZE HALT (⬅ CONT to resume).
- INPUT (ENTER to resume).

Using PROMPT...CONT for Input

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

To enter PROMPT in a program:

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

```
« ... "prompt-string" PROMPT ... »
```

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

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

To respond to PROMPT while running a program:

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

29 The message is displayed until you press ENTER or ATTN or until you update the status area (for example, by pressing ⬅ REVIEW).

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 25-19) to calculate its surface area. You don't have to enter data on the stack prior to program execution.

Arguments	Results
	1: *area*

Program:	**Comments:**
«	
"ENTER a, b IN ORDER:"	Puts the prompting string on the stack.
PROMPT	Displays the string in the status area, halts program execution, and returns calculator control to the keyboard.
TORSA	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.

29

[→] [CLR]
[VAR] TPRO

```
ENTER a, b IN ORDER:
4:
3:
2:
1:
TPRO | V2 | V1 | TST |TORSV|TORSA
```

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

8 [ENTER] 10

```
          HALT
{ HOME }
3:
2:
1:                        8
10◆
TPRO | V2 | V1 | TST |TORSV|TORSA
```

Continue the program.

[←] [CONT]

```
1:          355.305758439
TPRO | V2 | V1 | TST |TORSV|TORSA
```

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

Using DISP FREEZE HALT...CONT for Input

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

To enter DISP FREEZE HALT in a program:

1. Enter a string or other object to be displayed as a prompt.
2. Enter a number specifying the line to display it on.
3. Enter the DISP command (PRG CTRL menu).
4. Enter a number specifying the areas of the display to "freeze."
5. Enter the FREEZE command (PRG CTRL menu).
6. Enter the HALT command (PRG CTRL menu).

```
« ... prompt-object display-line DISP
      freeze-area FREEZE HALT ... »
```

29

DISP displays an object in a specified line of the display. DISP takes two arguments from the stack: an object from level 2, and a display-line number 1 through 7 from level 1. If the object is a string, it's displayed without the " " delimiters. The display created by DISP persists only as long as the program continues execution—if the program ends or is suspended by HALT, the calculator returns to the normal stack environment and updates the display. However, you can use FREEZE to retain the prompt display.

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

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

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

To respond to HALT while running a program:

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

Example: If you execute this program segment

```
« "ABC■DEF■GHI" CLLCD 1 DISP 3 FREEZE HALT »
```

the display looks like this:

(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 CTRL menu).
4. Enter OBJ→ (PRG OBJ 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. Program-entry mode is activated, the 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*.

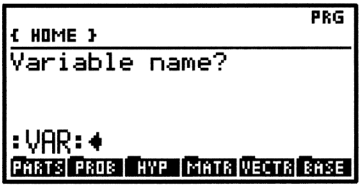
To respond to INPUT while running a program:

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 ATTN.
3. Press ENTER.

Example: If you execute this program segment

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

the display looks like this:



Example: The following program, *VSPH*, calculates the volume of a sphere. *VSPH* prompts for the radius of the sphere, then 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.

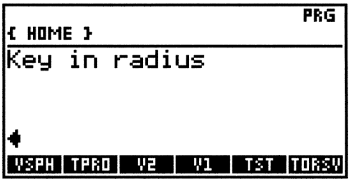
Arguments	Results
	1: *volume*

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*.

29

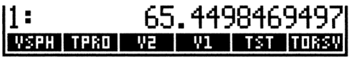
Execute *VSPH* to calculate the volume of a sphere of radius 2.5.

[VAR] VSPH



Key in the radius and continue program execution.

2.5 [ENTER]



To include INPUT options:

- Use a list (with { } delimiters) as the command-line argument for INPUT. The list can contain one or more of the following:
  - Command-line string (with " " delimiters).
  - Cursor position as a real number or as a list containing two real numbers.
  - Operating options ALG, α, or V.

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

{ "*command-line*" *cursor-position operating-options* }

"*command-line*" Specifies the content of the command line when the program pauses. Embedded newline characters produce multiple lines in the display. (If not included, the command line is blank.)

cursor-position Specifies the position of the cursor in the command line and its type. (If not included, an insert cursor is at the end of the command line.)

- A *real number n* specifies the *n*th character in the first row (line) of the command line. 0 specifies the end of the command-line string. A positive number specifies the *insert* cursor—a negative number specifies the *replace* cursor.
- A *list* {*row character*} specifies the row and character position. Row 1 is the first row (line) of the command line. Characters count from the left end of each row—character 0 specifies the end of the row. A positive row number specifies the *insert* cursor—a negative row number specifies the *replace* cursor.

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

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

To design the command-line string for INPUT:

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

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

To process the result string from INPUT:

- For simple input, use OBJ→ to convert the string into its corresponding objects.
- For sensitive input, use the V option for INPUT to check for valid objects, then use OBJ→ to convert the string into those objects. 29
- For special input, process the input as a string object, possibly extracting data as substrings.

Example: The program *VSPH* on page 29-6 uses an empty command-line string.

Example: The program *SSEC* on page 29-11 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 25-12 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 25-19) 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.

Arguments	Results
	1: *area*

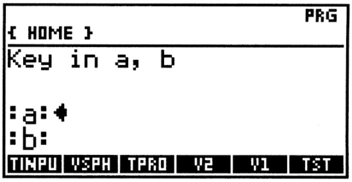
Program: | **Comments:**

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*.

29

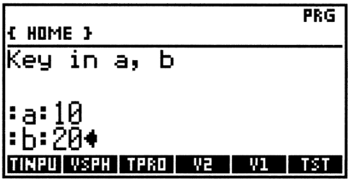
Execute *TINPUT* to calculate the surface area of a torus of inner radius $a = 10$ and outer radius $b = 20$.

(VAR) TINPU



Key in the value for a, press (▼) to move the cursor to the next prompt, then key in the value for b.

10 (▼) 20



Continue program execution.

(ENTER)

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

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

Arguments	Results
	2: "*first three digits*" 1: "*last four digits*"

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`	Copies the result string, then
`SWAP`	extracts the first three and last
`8 11 SUB`	four digits in string form.
`»`	
[ENTER] ['] SSEC [STO]	Stores the program in *SSEC*.

Beeping to Get Attention

29 The BEEP command lets you enhance an interactive program with audible prompting.

To enter BEEP in a program:

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

« ... *frequency duration* BEEP ... »

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

Example: The following edited version of *TPROMPT* sounds a 440-hertz, one-half-second tone at the prompt for data input.

Program:	Comments:
`«`	
`"ENTER a, b IN ORDER:"`	
`440 .5 BEEP`	Sounds a tone just before the prompt for data input.
`PROMPT`	
`TORSA`	
`»`	

Stopping for Keystroke Input

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

Using WAIT for Keystroke Input

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

To enter WAIT in a program:

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

WAIT takes the 0 or −1 from level 1, then suspends execution until a valid keystroke is executed. 29

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

When execution resumes, the three-digit key location number of the pressed key is left on the stack. This number indicates the row, column, and shift level of the key. (Key location numbers are explained under "Assigning User Keys" on page 15-6.)

To respond to WAIT while running a program:

- Press any valid keystroke. (A prefix key such as [↰] or [α] *by itself* is not a valid keystroke.)

Using KEY for Keystroke Input

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

To enter a KEY loop in a program:

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

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

The test-clause should normally cause the loop to repeat until a key is pressed. If a key is pressed, you can use comparison tests to check the value of the key number. (See "Using Indefinite Loop Structures" on page 27-10 and "Using Comparison Functions" on page 26-2.)

29

To respond to a KEY loop while running a program:

- Press any key. (A prefix key such as [⬅] or [α] *is* a valid key.)

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

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

Displaying Program Output

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

Labeling Output with Tags

To label a result with a tag:

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

```
« ... object tag →TAG ... »
```

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

Example: The following program *TTAG* is identical to *TINPUT*, except that it returns the result as `AREA:` *value*.

Program:	Comments:
`«`	
`"Key in a, b"`	
`{ ":a:■:b:" {1 0} V }`	
`INPUT OBJ→`	
`TORSA`	
`"AREA"`	Enters the tag (a string).
`→TAG`	Uses the program result and string to create the tagged object.
`»`	
[ENTER] ['] TTAG [STO]	Stores the program in *TTAG*.

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

[VAR] TTAG
1.5 [▼] 1.85
[ENTER]

```
1: AREA: 11.5721111603
TTAG SSEC TINPU VSPH TPRO V2
```

Labeling and Displaying Output as Strings

To label and display a result as a string:

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

« … *object* →STR *label* SWAP + *line* DISP … »

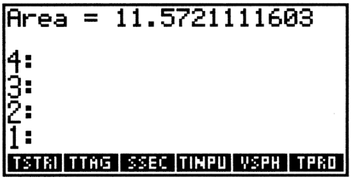
DISP displays a string without its " " delimiters.

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

Program:	Comments:
«	
"Key in a, b"	
{ ":a:■:b:" {1 0} V }	
INPUT OBJ→	
TORSA	
→STR	Converts the result to a string.
"Area = "	Enters the labeling string.
SWAP +	Swaps and adds the two strings.
CLLCD 1 DISP 1 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 is displayed in the status area.

[↱] [CLR]
[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 CTRL menu).

WAIT suspends execution for the (positive) 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 29-13.)

Summary of Data Input and Output Commands

Data Input Commands

Key	Programmable Command	Description
⬅ CONT	CONT	Restarts a halted program.
PRG CTRL (pages 1, 2, and 3):		
HALT	HALT	Halts program execution.
INPUT	INPUT	Suspends program execution for data input. Prevents stack operations while the program is paused.
PROM	PROMPT	Halts program execution for data input.
DISP	DISP	Displays an object in the specified line of the display.
WAIT	WAIT	Suspends program execution for a specified duration (seconds, level 1).
KEY	KEY	Returns a test result to level 1 and, if a key is pressed, the location of that key (level 2). See the next section, "Stopping for Keystroke Input."
BEEP	BEEP	Sounds a beep at a specified frequency (hertz, level 2) for a specified duration (seconds, level 1).
PRG DSPL (page 4):		
CLLCD	CLLCD	Blanks the display.
FREEZ	FREEZE	"Freezes" a specified area of the display so that it is not updated until a key press.

29

Using Menus with Programs

You can use menus with programs for different purposes:

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

To set up a built-in or library menu:

1. Enter the menu number.
2. Enter the MENU command (PRG CTRL menu).

To set up a custom menu:

29

1. Enter a list (with { } delimiters) or the name of a list defining the menu actions. (See "Using Custom Menus" on page 15-1.)
2. Activate the menu:
   - To save the menu as the CST menu, enter the MENU command (PRG CTRL menu).
   - To make the menu temporary, enter the TMENU command ([↱] [MODES] menu).

The menu isn't displayed until program execution halts.

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

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

To specify a particular *page* of a menu, enter the number as *m.pp*, where *m* is the menu number and *pp* is the page number (such as 35.02 for page 2 of the TIME menu). If page *pp* doesn't exist, page 1 is displayed (35 gives page 1 of the TIME menu).

Example: Enter `20 MENU` to get page 1 of the MODES menu. Enter `20.02 MENU` to get page 2 of the MODES menu.

To restore the previous menu:

- Execute 0 MENU.

To recall the menu number for the current menu:

- Execute the RCLMENU command (⮫ MODES menu).

Using Menus for Input

To display a menu for input in a program:

29

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

The program remains halted until it's resumed by a CONT command, such as by pressing ⮨ 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 3 of the MODES menu and prompts you to set the angle mode. After you press the menu key, you have to press ⮨ CONT to resume execution.

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

Example: The *PIE* program on page 31-40 assigns the CONT command to one key in a temporary menu.

Example: The *MNX* program on page 31-23 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.

List: | **Comments:**

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

['] LST [STO]	Stores the list in *LST*.

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 12.5 N. The program sets up the menu, then completes execution.

[VAR] WGT	ENGL | SI

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

SI

```
ENTER Wt in NEWTONS
4:
3:
2:
1:
ENGL SI
```

Key in the weight, then resume the program.

12.5 (⬑)(CONT)

```
1:          1.27420998981
ENGL SI
```

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

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

Program:	**Comments:**
`«`	
`DEG -15 SF -16 SF` `2 FIX`	Sets Degrees mode. Sets flags −15 and −16 to display complex numbers in polar form. Sets the display mode to 2 Fix.
`{`	Starts the custom menu list.

`{ "E" { « 'E' STO »`
`« I Z * DUP 'E' STO`
`"E: " SWAP + CLLCD`
`1 DISP 1 FREEZE »`
`« E » } }`
	Builds menu key 1 for E. Unshifted action: stores the object in *E*. Left-shift action: calculates $I \times Z$, stores it in *E*, and displays it with a label. Right-shift action: recalls the object in *E*.

`{ "I" { « 'I' STO »`
`« E Z / DUP 'I' STO`
`"I: " SWAP + CLLCD`
`1 DISP 1 FREEZE »`
`« I » } }`
	Builds menu key 2.

`{ "Z" { « 'Z' STO »`
`« E I / DUP 'Z' STO`
`"Z: " SWAP + CLLCD`
`1 DISP 1 FREEZE »`
`« Z » } }`
	Builds menu key 3.

`}`	Ends the list.
`MENU`	Displays the custom menu.
`»`	
[ENTER] ['] EIZ [STO]	Stores the program in *EIZ*.

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

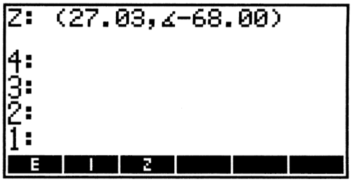
[→] [CLR] [VAR] `EIZ`

Key in the voltage value.

[←] [( )] 10 [→] [∠] 0

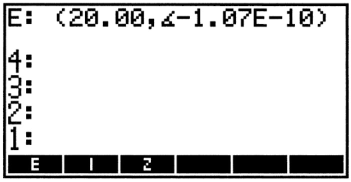
Store the voltage value. Then key in and store the current value. Solve for the impedance.

E
[⬅] [( )] .37 [➡] [∡] 68 I
[⬅] Z



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

[➡] I
2 [×]
I
[⬅] E



Press [⬅] [MODES] STD and [➡] [POLAR] to restore Standard and Rectangular modes.

29

Turning Off the HP 48 from a Program

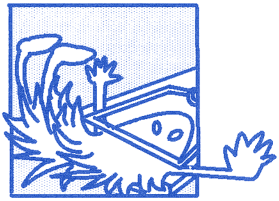
To turn off the calculator in a program:

- Execute the OFF command (PRG CTRL menu).

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

30

Error Trapping



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

In a program, the same thing happens, but program execution is also aborted. If you anticipate error conditions, your program can process them without interrupting execution.

For simple programs, you can run the program again if it stops with an error. For other programs, you can design them to *trap* errors and continue executing. You can also create user-defined errors to trap certain conditions in programs.

Causing and Analyzing Errors

Many conditions are automatically recognized by the HP 48 as error conditions—and they're automatically treated as errors in programs. A command with an improper argument or an improper number of arguments causes an error in a program. An out-of-range result can cause an error. An invalid calculator condition can cause an error.

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

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

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

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

1. Enter the error number (as a binary integer or real number) for the error.
2. Enter the DOERR command (PRG CTRL 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 is displayed.

To analyze an error in a program:

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

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

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

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

The following table summarizes error trapping commands.

Error Trapping Commands

Key	Programmable Command	Description
(PRG) CTRL (page 3):		
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 IFFER structure, DOERR displays the message and abandons program execution. (For 0 in level 1, abandons execution without updating the error number or message—like (ATTN).)
ERRN	ERRN	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.

30

Trapping Errors

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

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

The IFERR...THEN...END Structure

The syntax for this structure is

« ... `IFERR` *trap-clause* `THEN` *error-clause* `END` ... »

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

30

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

- Press [PRG] BRCH [◄][PREV] [◄] IFERR.

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

```
« WHILE DUP TYPE 3 == REPEAT Σ+ END »
```

In the following revised version, the program simply attempts to add the level 1 object to the statistics matrix until an error occurs. Then, it "gracefully" ends by displaying the message `DONE`.

Program:	**Comments:**
«	
IFERR	Starts the trap-clause.
WHILE	The WHILE structure repeats
1	indefinitely, adding the vectors
REPEAT	and arrays to the statistics matrix
Σ+	until an error occurs.
END	
THEN	Starts the error clause. If an error
"DONE" 1 DISP	occurs in the WHILE structure,
1 FREEZE	displays the message DONE in the status area.
END	Ends the error structure.
»	

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

30

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) CTRL (←)(PREV) (→) IFERR.

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

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

31

More Programming Examples

The programs in this chapter demonstrate programming concepts introduced in the previous chapters. Some new concepts are also introduced. The programs are intended to both improve your programming skills and provide supplementary functions for your calculator.

At the end of each program, the *checksum* and the program *size* in bytes are listed. The checksum is a binary integer that uniquely identifies the program based on its contents. To verify that you've keyed the program in correctly, execute the BYTES command (⬅ MEMORY BYTES) with the program *name* in level 1. The checksum for the program is returned to level 2, and its size in bytes is returned to level 1. (If you execute BYTES with the program *object* in level 1, before storing the program in its name, you'll get a different byte count returned to level 1.)

These programs are also included in the online information of the Program Development Link, software for developing HP 48 programs on computers. Using this software, you can cut and paste these programs from the online information, then load them into the HP 48 via its serial port.

The examples in this chapter assume the HP 48 is in its initial, default condition—they assume you haven't changed any of the HP 48 operating modes. (To reset the calculator to this condition, see "If Things Go Wrong" on page A-1.)

Fibonacci Numbers

This section includes three programs—two demonstrate an approach to the following problem:

Given an integer n, calculate the nth Fibonacci number F_n, where:

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

- *FIB1* is a user-defined function that is defined *recursively*—its defining procedure contains its own name. *FIB1* is short.
- *FIB2* is a user-defined function with a definite loop. It's longer and more complicated than *FIB1*, but it's faster.

The third program, *FIBT*, calls both *FIB1* and *FIB2*, then calculates the execution time of each subprogram.

FIB1 (Fibonacci Numbers, Recursive Version)

31

Arguments	Results
1: n	1: F_n

Techniques

- 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. (*FIB2* uses the conditional structure IF...THEN...ELSE...END.)
- 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}.

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

Checksum: # 41467d
Bytes: 113.5

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

First calculate F_6.

[VAR]
6 FIB1

Next calculate F_{10} using algebraic syntax.

['] FIB1 [⬅][()] 10 [EVAL]

FIB2 (Fibonacci Numbers, Loop Version)

31

Arguments	Results
1: n	1: F_n

Techniques

- 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 F_i values.

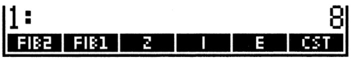
Program:	Comments:
`«`	
`  → n`	Creates a local variable structure.
`  «`	
`    IF n 1 ≤`	If $n \le 1$
`    THEN n`	then $F_n = n$
`    ELSE`	otherwise ...
`      0 1`	Puts F_0 and F_1 on the stack.
`      2 n`	From 2 to n does the following loop:
`      START`	
`        DUP`	Copies the latest F (initially F_1).
`        ROT`	Gets the previous F (initially F_0).
`        +`	Calculates the next F (initially F_2).
`      NEXT`	Repeats the loop.
`      SWAP DROP`	Drops F_{n-1}.
`    END`	Ends the ELSE clause.
`  »`	Ends the defining procedure.
`»`	
[ENTER] ['] FIB2 [STO]	Stores the program in *FIB2*.

Checksum: # 51820d
Bytes: 89

Example: Calculate F_6 and F_{10}.

Calculate F_6.

[VAR]
6 `FIB2`



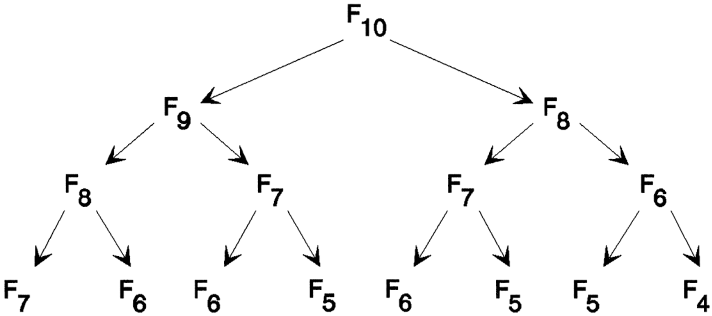
Calculate F_{10}.

10 `FIB2`

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.

The diagram below shows the beginning steps of *FIB1* calculating F_{10}. Note the number of intermediate calculations: 1 in the first row, 2 in the second row, 4 in the third row, and 8 in the fourth row.



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

Arguments	Results
	3: F_n
	2: FIB1 TIME: z
1: n	1: FIB2 TIME: z

Techniques

- 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.
- Interactive programming. *FIBT* tags each execution time with a descriptive message.

Required Programs

- *FIB1* (page 31-2) calculates F_n using recursion.
- *FIB2* (page 31-3) calculates F_n using looping.

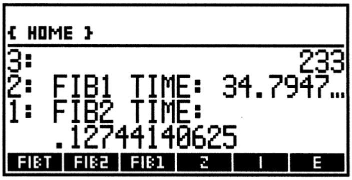
Program:	Comments:
«	
DUP TICKS SWAP FIB1 SWAP TICKS SWAP	Copies n, then executes *FIB1*, recording the start and stop time.
- B→R 8192 /	Calculates the elapsed time, converts it to a real number, and converts that number to seconds. Leaves the answer returned by *FIB1* in level 2.
"FIB1 TIME" →TAG	Tags 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: # 22248d
Bytes: 135

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

Select the VAR menu and do the calculation.

VAR
13 FIBT



F_{13} is 233. *FIB2* takes fewer seconds to execute than *FIB1*. (Your results will differ depending on the contents of memory and other factors.)

Displaying a Binary Integer

This section contains three programs:

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

PAD (Pad with Leading Spaces)

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

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

Arguments	Results
`1:` *object*	`1:` " *object*"

Techniques

- WHILE...REPEAT...END (indefinite loop). The WHILE clause contains a test that determines whether to execute the REPEAT clause and test again (if true) or to skip the REPEAT clause and exit (if false).

- String operations. *PAD* demonstrates how to convert an object to string form, count the number of characters, and combine two strings.

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

31

Checksum: # 38912d
Bytes: 61.5

PAD is demonstrated in the program *BDISP*.

PRESERVE (Save and Restore Previous Status)

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

Arguments	Results
1: « *program* »	1: *result of program*
1: '*program name*'	1: *result of program*

Techniques

- RCLF and STOF. *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 remove the binary integer from the stack briefly; 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 execution of the program on the stack and to restore flags. DOERR shows the error if one occurs.

Program:	Comments:
`«`	
`RCLF`	Recalls the list of two 64-bit binary integers representing the status of the 64 system flags and 64 user flags.
`→ f`	Stores the list in local variable f.
`«`	Begins the defining procedure.
`IFERR`	Starts the error trap.
`EVAL`	Executes the program placed on the stack as the level 1 argument.
`THEN`	If the program caused an error,
`f STOF ERRN DOERR`	restores flags, shows the error, and aborts execution.
`END`	Ends the error routine.
`f STOF`	Puts the list back on the stack, then restores the status of all flags.
`»`	Ends the defining procedure.
`»`	
[ENTER] ['] PRESERVE [STO]	Stores the program in *PRESERVE*.

Checksum: # 7284d
Bytes: 71

PRESERVE is demonstrated in the program *BDISP*.

31

BDISP (Binary Display)

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

Arguments	Results
1: # *n*	1: # *n*
1: *n*	1: *n*

Techniques

31

- 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, LASTARG must be enabled to return the argument (the binary number) to the stack. *BDISP* clears flag −55 to enable the LASTARG recovery feature.
- 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. It is created by the FOR...NEXT program structure (rather than by a → command) and it is 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; it is simply placed on the stack and may 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*.
  - □ There are four program arguments that "customize" the action of the loop. Each program argument 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* (page 31-7) expands a string to 22 characters so that DISP shows it right-justified.
- *PRESERVE* (page 31-8) stores the current status, executes the main nested program and restores the status.

Program:	Comments:
`«`	
`«`	Begins the main nested program.
`DUP`	Makes a copy of n.
`-55 CF`	Clears flag −55 to enable LASTARG.
`IFERR`	Begins error trap.
`R→B`	Converts n to a binary integer.
`THEN`	If an error occurred, do nothing
`END`	(no commands in the THEN clause).
`→ n`	Creates a local variable n and
`«`	begins the defining program.
`CLLCD`	Clears the display.
`« BIN »`	Nested program for BIN.
`« OCT »`	Nested program for OCT.
`« DEC »`	Nested program for DEC.
`« HEX »`	Nested program for HEX.

31

Program:	Comments:
`1 4`	Sets the counter limits.
`FOR j`	Starts the loop with counter *j*.
`EVAL`	Executes one of the nested base programs (initially for HEX).
`n →STR`	Makes a string showing *n* in the current base.
`PAD`	Pads the string to 22 characters.
`j DISP`	Displays the string in the *j*th line.
`NEXT`	Increments *j* and repeats the loop.
`»`	Ends the defining program.
`3 FREEZE`	Freezes the status and stack areas.
`»`	Ends the main nested program.
`PRESERVE`	Stores the current flag status, executes the main nested program, and restores the status.
`»`	
[ENTER] ['] BDISP [STO]	Stores the program in *BDISP*.

31

Checksum: # 18055d
Bytes: 191

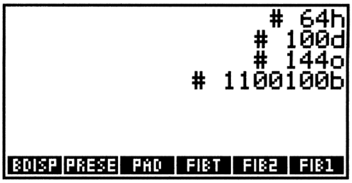
Example: Switch to DEC base, display #100 in all bases, and check that *BDISP* restored the base to DEC.

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

[↱] [CLR]
[MTH] BASE
DEC
[↱] [#] 100 [ENTER]

Execute *BDISP*.

[VAR] BDISP

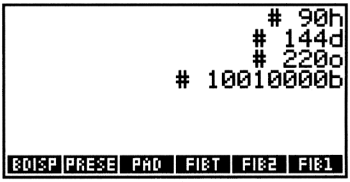


Return to the normal stack display and check the current base.

[ATTN]
[MTH] BASE

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

[VAR]
144 BDISP



Press [ATTN] to return to the stack display.

Median of Statistics Data

This section contains three programs:

- *SORT* orders the elements of a list.
- *LMED* calculates the median of a sorted list.
- *MEDIAN* uses *SORT* and *LMED* to calculate the median of the current statistics data.

SORT (Sort a List)

SORT sorts a list of real numbers into ascending order.

Arguments	Results
1: { *list* }	1: { *sorted list* }

31

Techniques

- Bubble sort. Starting with the first and second numbers in the list, *SORT* compares adjacent numbers and moves the larger number toward the end of the list. This process is done once to move the largest number to the last position in the list, then again to move the next largest to the next-to-last position, and so on.
- Nested definite loops. The outer loop controls the stopping position each time the process is done; the inner loop runs from 1 to the stopping position each time the process is done.
- FOR...STEP and FOR...NEXT (definite loops). *SORT* uses two counters: −1 STEP decrements the counter for the outer loop each iteration; NEXT increments the counter for the inner loop by 1 each iteration.

Program:	Comments:
«	
DUP SIZE 1 - 1 FOR j	From the next-to-last position to the first position, begins the outer loop with counter *j*.
1 j FOR k	From the first position to the *j*th position, begins the inner loop with counter *k*.
DUP k GETI 3 ROLLD GET → n1 n2	Gets the *k*th and *k+1*st numbers in the list and stores them in local variables n_1 and n_2.
«	Begins the defining procedure (a program) for the local variable structure.
IF n1 n2 > THEN k n2 PUTI n1 PUT END	If the two numbers are in the wrong order, puts them back in the opposite positions.
»	Ends the defining procedure.
NEXT	Increments *k* and repeats the inner loop.
-1 STEP	Decrements *j* and repeats the outer loop.
»	
ENTER ' SORT STO	Stores the program in *SORT*.

31

Checksum: # 51893d
Bytes: 141.5

Example: Sort the list { 8 3 1 2 5 }.

Select the VAR menu, key in the list, and execute *SORT*.

VAR
← {} 8 SPC 3 SPC
1 SPC 2 SPC 5 ENTER
SORT

LMED (Median of a List)

Given a sorted list, *LMED* returns the median. If the list contains an odd number of elements, the median is the value of the center element. If the list contains an even number of elements, the median is the average value of the elements just above and below the center.

Arguments	Results
1: { *sorted list* }	1: *median of sorted list*

Techniques

- FLOOR and CEIL. For an integer, FLOOR and CEIL both return that integer; for a noninteger, FLOOR and CEIL return successive integers that bracket the noninteger.

31

Program:	Comments:
«	
DUP SIZE	Copies the list, then finds its size.
1 + 2 /	Calculates the center position in the list (fractional for even-sized lists).
→ p	Stores the center position in local variable *p*.
«	Begins the defining procedure.
DUP	Makes a copy of the list.
p FLOOR GET	Gets the number at or below the center position.
SWAP	Moves the list to level 1.
p CEIL GET	Gets the number at or above the center position.
+ 2 /	Calculates the average of the two numbers.
»	Ends the defining procedure.
»	
[ENTER] ['] LMED [STO]	Stores the program in *LMED*.

Checksum: # 3682d
Bytes: 77

Example: Calculate the median of the list you sorted using *SORT*. (Put the list on the stack, if necessary.)

[←][{}] 1 2 3 5 8 [ENTER]
[VAR] LMED

```
1:                              3
LMED SORT BDISP PRESE PAD  FIBT
```

MEDIAN (Median of Statistics Data)

MEDIAN returns a vector representing the medians of the columns of the statistics data.

Arguments	Results
1:	1: [x_1 x_2 ... x_m]

Techniques

- 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 then combines them into a list. From this list the median is calculated using *SORT* and *LMED*.

  The median for the *m*th column is calculated first, and the median for the first column is calculated last, so as each median is calculated, it is moved to the stack level above the previously calculated medians.

  After all medians are calculated and positioned correctly 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 Programs

- *SORT* (page 31-14) arranges a list in ascending order.
- *LMED* (page 31-16) calculates the median of a sorted list.

31

Program:	Comments:
`«`	
`RCLΣ`	Puts a copy, s, 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. Drops the list size.
`→ s n m`	Creates local variables for s, n, and m.
`«`	Begins the defining procedure.
`'ΣDAT' TRN`	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:
`Σ-`	Extracts the last row in ΣDAT. Initially this is the mth row, which corresponds to the mth column in the original ΣDAT. (To key in the Σ− command, press [←] [STAT] [←] [Σ+].)
`OBJ→ DROP`	Puts the row elements on the stack. Drops the index list { n }.
`n →LIST`	Makes an n-element list.
`SORT`	Sorts the list.
`LMED`	Calculates the median of the list.
`j ROLLD`	Moves the median to the proper stack level.
`NEXT`	Increments j and repeats the loop.

Program:	**Comments:**
`m →ARRY`	Combines all the medians into an m-element vector.
`s STOΣ`	Restores *ΣDAT* to its previous value.
`»`	Ends the defining procedure.
`»`	
ENTER ' MEDIAN STO	Stores the program in *MEDIAN*.

Checksum: # 3947d
Bytes: 136

Example: Calculate the median of the following data.

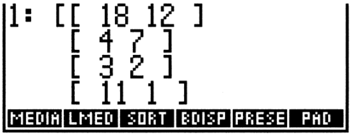
$$\begin{bmatrix} 18 & 12 \\ 4 & 7 \\ 3 & 2 \\ 11 & 1 \\ 31 & 48 \\ 20 & 17 \end{bmatrix}$$

31

There are two columns of data, so *MEDIAN* will return a two-element vector.

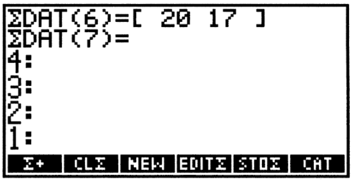
Enter the matrix.

→ MATRIX
18 ENTER 12 ENTER ▼
4 ENTER 7 ENTER
3 ENTER 2 ENTER
11 ENTER 1 ENTER
31 ENTER 48 ENTER
20 ENTER 17 ENTER
ENTER



Store the matrix in *ΣDAT*.

→ CLR
← STAT CLΣ Σ+



Calculate the median. The medians are 14.5 for the first column and 9.5 for the second column.

(VAR) MEDIA

```
1:          [ 14.5 9.5 ]
ΣDAT MEDIA LMED SORT BDISP PRESE
```

Expanding and Collecting Completely

This section contains two programs:

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

31

MULTI (Multiple Execution)

Given an object and a program that acts on the object, *MULTI* applies the program to the object repeatedly until the object is unchanged.

Arguments	Results
2: *object*	
1: « *program* »	1: *resulting object*

Techniques

- DO...UNTIL...END (indefinite loop). The DO clause contains the steps to be repeated; the UNTIL clause contains the test that determines whether to repeat both clauses again (if false) or to exit (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.

Program:	Comments:
`«`	
`→ p`	Creates a local variable *p* containing 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: # 34314d
Bytes: 56

MULTI is demonstrated in the next programming example.

EXCO (Expand and Collect Completely)

Given an algebraic object, *EXCO* executes EXPAN repeatedly until the algebraic doesn't change, then executes COLCT repeatedly until the algebraic doesn't change. In some cases the result will be a number.

31

Expressions with many products of sums or with powers can take many iterations of EXPAN to expand completely, resulting in a long execution time for *EXCO*.

Arguments	Results
1: '*algebraic*'	1: '*algebraic*'
1: '*algebraic*'	1: *z*

Techniques

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

Required Programs

- *MULTI* (page 31-20) repeatedly executes the programs that *EXCO* provides as arguments.

31

Program:	Comments:
«	
« EXPAN »	Puts a program on the stack as the level 1 argument for *MULTI*. The program executes the EXPAN command.
MULTI	Executes EXPAN until the algebraic object doesn't change.
« COLCT »	Puts another program on the stack for *MULTI*. The program executes the COLCT command.
MULTI	Executes COLCT until the algebraic object doesn't change.
»	
[ENTER] ['] EXCO [STO]	Stores the program in *EXCO*.

Checksum: # 48008d
Bytes: 65.5

Example: Expand and collect completely the expression:

$$3x(4y+z)+(8x-5z)^2$$

Enter the expression.

['] 3 [×] X [×]
[←] [( )] 4 [×] Y [+] Z [▶] [+]
[←] [( )] 8 [×] X [−] 5 [×] Z
[▶] [y^x] 2
[ENTER]

```
1: '3*X*(4*Y+Z)+(8*X-5
    *Z)^2'
EXCO MULTI ΣDAT MEDIA LMED SORT
```

Select the VAR menu and start the program.

[VAR] EXCO

```
1: '64*X^2+12*X*Y-77*X
    *Z+25*Z^2'
EXCO MULTI ΣDAT MEDIA LMED SORT
```


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)

Given an array on the stack, *MNX* finds the minimum or maximum element in the array.

Arguments	Results
	2: [[*array*]]
1: [[*array*]]	1: *z* (min or max element)

Techniques

- 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—it determines:
  - 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 for making a choice. *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.

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 to level 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`	Begins the DO test-clause.
`-64 FS?`	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: # 57179d
Bytes: 210.5

Example: Find the maximum element of the following matrix:

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

Enter the matrix.

(↱)(CLR) (↱)(MATRIX)
12 (ENTER) 56 (ENTER) (▼)
45 (ENTER) 1 (ENTER)
9 (ENTER) 14 (ENTER)
(ENTER)

```
1: [[ 12 56 ]
    [ 45 1 ]
    [ 9 14 ]]
MNX  EXCO MULTI ΣDAT MEDIA LMED
```

Select the VAR menu and execute *MNX*.

(VAR) MNX

```
Sort for MAX or MIN?
2:
1: [[ 12 56 ]
    [ 45 1 ]
    [ 9 14 ]]
MAX  MIN
```

Find the maximum element.

MAX

```
2: [[ 12 56 ] [ 45 1…
1:                 56
MNX  EXCO MULTI ΣDAT MEDIA LMED
```

MNX2 (Minimum or Maximum Element—Version 2)

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

Arguments	Results
	`2:` `[[`*array*`]]`
`1:` `[[`*array*`]]`	`1:` *z* (min or max element)

Techniques

- FOR...NEXT (definite loop). The initial counter value is 1. The final counter value is $nm - 1$ where nm is the number of elements in the array. The loop-clause contains the sort instructions.
- User flag for logic control. *User* flag 10 defines the sort: When flag 10 is set, *MNX2* finds the maximum element; when flag 10 is clear, it finds the minimum element. *You* determine the status of flag 10 at the beginning of the program.
- Nested conditional. An IF...THEN...END conditional is nested in the FOR...NEXT loop—it determines:
  - Whether to maintain the current minimum or maximum element, or make the current element the new minimum or maximum.
  - The sense of the comparison of elements (either $<$ or $>$) based on the status of flag 10.
- Logical function. *MNX2* executes XOR (*exclusive OR*) to test the combined state of the relative value of the two elements and the status of flag 10. 31
- Custom menu for making a choice. *MNX2* builds a custom menu that lets you choose whether to sort for the minimum or maximum element. Key 1, labeled MAX, sets flag 10. Key 2, labeled MIN, clears flag 10.

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`
`"Sort for MAX or MIN?"`
` PROMPT`
	Displays the temporary menu and a 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.

Program:	**Comments:**
`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: # 12277d
Bytes: 200.5

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

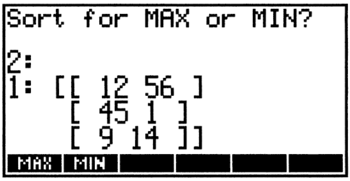
$$\begin{bmatrix} 12 & 56 \\ 45 & 1 \\ 9 & 14 \end{bmatrix}$$

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

(↱) (MATRIX)
12 (ENTER) 56 (ENTER) (▼)
45 (ENTER) 1 (ENTER)
9 (ENTER) 14 (ENTER)
(ENTER)

Select the VAR menu and execute *MNX2*.

(VAR) MNX2



Find the minimum element.

MIN

Verification of Program Arguments

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

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

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

NAMES (Check List for Exactly Two Names)

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

31

Arguments	Results
`1: {` *valid list* `}`	`1:`
`1: {` *invalid list* `}`	*status-area error message* `1:`

Techniques

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

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

Checksum: # 40666d
Bytes: 141.5

NAMES is demonstrated in program *VFY*.

VFY (Verify Program Argument)

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

Arguments	Results
1: '*name*'	1: '*name*'
1: { *valid list* }	1: { *valid list* }
1: { *invalid list* }	*status-area error message* 1: { *invalid list* }
1: *invalid object*	*status-area error message* 1: *invalid object*

Techniques

31

- Utility programs. *VFY* by itself has little use. However, it can be used (with minor modifications) by other programs to verify that specific object types are valid arguments.
- CASE...END (case structure). *VFY* uses a case structure to determine if the argument is a list or a name.
- Structured programming. If the argument is a list, *VFY* calls *NAMES* to verify that the list is valid.
- Local variable structure. *VFY* stores its argument in a local variable so that it may be passed to *NAMES* if necessary.
- Logical operator. *VFY* uses NOT to display an error message.

Required Programs

- *NAMES* (page 31-30) verifies that a list argument contains exactly two names.

Program:	Comments:
`«`	
`DUP`	Copies the original argument to leave on the stack.
`DTAG`	Removes any tags from the argument for subsequent testing.
`→ argm`	Stores the argument in local variable *argm*.
`«`	Begins the defining procedure.
`CASE`	Begins the case structure.
`argm TYPE 5 SAME THEN argm NAMES END`	Tests if the argument is a list. If so, puts the argument back on the stack and calls *NAMES* to verify that the list is valid, then leaves the CASE structure.
`argm TYPE 6 SAME NOT THEN "Not name or list" DOERR END`	Tests if the argument is *not* a name. If so, displays an error message and aborts execution.
`END`	Ends the CASE structure.
`»`	Ends the defining procedure.
`»`	
ENTER ' VFY STO	Enters the program, then stores it in *VFY*.

31

```
«
 DUP
 DTAG
 → argm
 «
  CASE
    argm TYPE 5 SAME
   THEN
    argm NAMES
   END
    argm TYPE 6 SAME NOT
   THEN
    "Not name or list"
    DOERR
   END
  END
 »
»
```

Checksum: # 36796d
Bytes: 139.5

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

⮫ CLR
' PAT ENTER
VAR VFY

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

['] DIANA [ENTER]
['] TED [ENTER]
3 [PRG] OBJ →LIST

```
1:    { PAT DIANA TED }
OBJ→ EQ→ →ARR →LIST →STR →TAG
```

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

[VAR] VFY

```
Illegal list size
4:
3:
2:
1:    { PAT DIANA TED }
VFY NAME MNX2 MNX EXCO MULTI
```

31

Bessel Functions

The real and imaginary parts of the Bessel function $J_n(xe^{3\pi i/4})$ are denoted $\mathrm{Ber_n}(x)$ and $\mathrm{Bei_n}(x)$. When $n = 0$,

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

User-defined function *BER* calculates $\mathrm{Ber}(x)$.

Arguments	Results
1: z	1: $\mathrm{Ber}(z)$

Techniques

- 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 takes numeric or symbolic arguments from the stack or in algebraic syntax. Because *BER* uses a DO...UNTIL...END loop, its defining procedure is a *program*. (Loop structures are not allowed in algebraic expressions.) Therefore, unlike a user-defined function, *BER* is not differentiable.

- DO...UNTIL...END loop (indefinite loop with counter). Successive terms in the series are calculated with a counter variable. When the new term does not change the series value 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 the storing and recalling of key parameters.

Program:	**Comments:**
`«`	
`→ x`	Creates local variable x.
`«`	Begins outer defining procedure.
`'x/2' →NUM 2 1` `→ xover2 j sum`	Enters $x/2$, the first counter value, and the first term of the series, then creates local variables.
`«`	Begins inner defining procedure.
`DO`	Begins the loop.
`sum` `'sum+(-1)^(j/2)*` `xover2^(2*j)/SQ(j!)'` `EVAL`	Recalls the old sum and calculates the new sum.
`2 'j' STO+`	Increments the counter.
`DUP 'sum' STO`	Stores the new sum.
`UNTIL`	Ends the loop clause.
`==`	Tests the old and new sums.
`END`	Ends the loop.
`sum`	Recalls the sum.
`»`	Ends inner defining procedure.
`»`	Ends outer defining procedure.
`»`	
[ENTER] ['] BER [STO]	Stores the program in *BER*.

Checksum: # 36388d
Bytes: 200.5

Example: Calculate Ber(3).

VAR
3 BER

```
1:          -.2213802496
 BER  VFY  NAME MNX2 MNX  EXCO
```

Calculate Ber(2) in algebraic syntax.

' BER ⬅ ( ) 2
EVAL

```
1:          .751734182714
 BER  VFY  NAME MNX2 MNX  EXCO
```

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.

31 ■ *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 each graphics object in a list.

■ *TSA* displays 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. Assumes Radians mode is active.

Arguments	Results
1:	1:

Techniques

■ Programmatic use of PLOT commands to build and display a graphics object.

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: # 1971d
Bytes: 91.5

31

SETTS (Superimposing Taylor's Polynomials)

SETTS superimposes successive Taylor's polynomials on a sine curve and stores each graphics object in a list.

Arguments	Results
1:	1:

Techniques

- Structured programming. *SETTS* calls *SINTP* to build a sine curve and convert it to a graphics object.
- FOR...STEP (definite) loop. *SETTS* calculates successive Taylor's polynomials for the sine function in a definite loop. The loop counter serves as the value of the order of each polynomial.
- Programmatic use of PLOT commands. *SETTS* draws a plot of each Taylor's polynomial.
- Manipulation of graphics objects. *SETTS* converts each Taylor's polynomial plot into a graphics object. Then it executes + to combine each graphics object with the sine curve stored in *SINT*, creating nine new graphics objects, each the superposition of a

Taylor's polynomial on a sine curve. *SETTS* then puts the nine new graphics objects, and the sine curve graphics object itself, in a list.

Program:	**Comments:**
`«`	
`SINTP`	Plots a sine curve and stores the graphics object in *SINT*.
`17 1 FOR n`	Sets the range for the FOR loop using local variable *n*.
`'SIN(X)' 'X' n TAYLR STEQ ERASE DRAW`	Plots the Taylor's polynomial of order *n*.
`PICT RCL SINT +`	Returns the plot to the stack as a graphics object and executes + to superimpose the sine plot from *SINT*.
`-2 STEP`	Decrements the loop counter *n* by 2 and repeats the loop.
31 `SINT 10 →LIST 'TSL' STO`	Puts the sine curve graphics object on the stack, then builds a list containing it and the nine graphics objects created in the loop. Stores the list in *TSL*.
`»`	
[ENTER] ['] SETTS [STO]	Stores the program in *SETTS*.

Checksum: # 57905d
Bytes: 138.5

TSA (Animating Taylor's Polynomials)

TSA displays in succession each graphics object created in *SETTS*.

Arguments	Results
1:	1:

Techniques

- Passing a global variable. Because *SETTS* takes several minutes to execute, *TSA* does not call *SETTS*. Instead, you must first execute

SETTS to create the global variable *TSL* containing the list of graphics objects. *TSA* simply executes that global variable to put the list on the stack.

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

Program:	**Comments:**
«	
TSL OBJ→	Puts the list *TSL* on the stack and converts it to 10 graphics objects and the list count.
1 SWAP FOR s ERASE →LCD 1 WAIT	For *s* from 1 to 10, clears the display, displays the next graphics object, and waits for 1 second.
NEXT	
»	
[ENTER] ['] TSA [STO]	Stores the program in *TSA*.

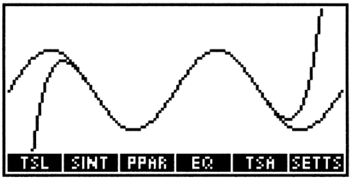
31

Checksum: # 39562d
Bytes: 51

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

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

[←][RAD] (if necessary)
[VAR] SETTS
TSA



Press [←][RAD] to restore Degrees mode.

Programmatic Use of Statistics and Plotting

Program *PIE* prompts for single variable data, stores that data in the statistics matrix *ΣDAT*, then draws a labeled pie chart that shows each data point as a percentage of the total.

Arguments	Results
1:	1:

Techniques

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

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 *ΣDAT*, key 3 clears *ΣDAT*, 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.
(66,32) 20 0 6.28 ARC	Draws the circle.
PICT RCL →LCD	Displays the empty circle.
RCLΣ TOT /	Recalls the statistics data matrix, computes totals, and calculates the proportions.
DUP 100 *	Converts the proportions to percentages.
→ prcnts	Stores the percentage matrix in *prcnts*.
«	
2 π →NUM * * 0	Multiplies the proportion matrix by 2π, and enters initial angle (0).

Program:	Comments:
`→ prop angle`	Stores the angle matrix in *prop* and angle in *angle*.
`«`	
`prop SIZE OBJ→` `DROP SWAP`	Sets up 1 to *m* as loop counter range.
`FOR n`	Begins loop-clause.
`(66,32) prop n GET` `'angle' STO+`	Puts the center of the circle on the stack, then gets the *n*th value from the proportion matrix and adds it to *angle*.

`angle COS angle SIN`
`R→C 20 * OVER +`
`LINE`
	Computes the endpoint and draws the line for the *n*th slice.

`PICT RCL`	Recalls *PICT* to the stack.

`angle prop n GET`
`2 / - DUP DUP`
`COS SWAP SIN R→C`
`26 * (66,32) +`
	For labeling the slice, computes the midpoint of the arc of the slice.

`SWAP` `CASE`	Starts the CASE structure to test *angle* and determine the offset value for the label.

`DUP 1.5 ≤`
`THEN`
`DROP`
`END`
	From 0 to 1.5 radians, doesn't offset the label.

`DUP 4.4 ≤`
`THEN`
`DROP 15 -`
`END`
	From 1.5 to 4.4 radians, offsets the label 15 user units left.

`5 <`
`THEN`
`(3,2) +`
`END`
	From 4.4 to 5 radians, offsets the label 3 units right and 2 units up.

`END`	Ends the CASE structure.

Program:	Comments:
`prcnts n GET`	Gets the *n*th value from the
`1 RND`	percentage matrix, rounds it to
`→STR "%" +`	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`	Displays the updated plot.
`NEXT`	Ends the loop structure.
`{ } PVIEW`	Displays the finished plot.
`»`	
`»`	
`flags STOF`	Restores the original flag status.
`» 0 MENU`	Restores the previous menu. (The user must first press [ATTN] to clear the plot.) 31
`»`	
[ENTER] ['] PIE [STO]	Stores the program in *PIE*.

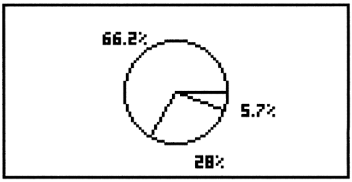
Checksum: # 1177d
Bytes: 765

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.

[→] [CLR]
[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 (ATTN) to return to the stack display.

Trace Mode

Programs *αENTER* and *βENTER* provide "trace mode" for the HP 48 using an external printer. To turn on "trace mode," set flag −63 and activate User mode. To turn off "trace mode," clear flag −63 or turn off User mode.

Techniques

- Vectored ENTER. Setting flag −63 and activating User mode turns on vectored ENTER. When vectored ENTER is turned on and variable *αENTER* exists, the command-line text is put on the stack as a string and *αENTER* is evaluated. Then, if variable *βENTER* exists, the command that triggered the command-line processing is put on the stack as a string and *βENTER* is evaluated.

Program:	**Comments:**
«	
PR1 OBJ→	Prints the command line text, then converts the string to objects and evaluates it.
»	
(') αENTER (STO)	Stores the program in *αENTER*. (Press (↱)A to type α. You *must* use this name.)

Checksum: # 51789d
Bytes: 25.5

Program:	**Comments:**
«	
PR1 DROP PRSTC	Prints the command that caused the processing, then drops it and prints the stack in compact form.
»	
['] βENTER [STO]	Stores the program in *βENTER*. (Press [→] B to type β. You *must* use this name.)

Checksum: # 37631d
Bytes: 28

31

Inverse-Function Solver

Program *ROOTR* 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).

Arguments	Results
3: *function name*	
2: *y-value*	
1: *x-guess*	1: *x-value*

Techniques

- 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.

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

31 Checksum: # 13007d
Bytes: 163

Example: Assume you often work with the expression $3.7x^3 + 4.5x^2 + 3.9x + 5$ and have created the program *X→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. To find the value of x for which the function equals 599.5 (using a guess in the vicinity of 1), enter the name `'X→FX'`, the y-value `599.5`, and the guess `1`—then press [VAR] `ROOTR`. The program returns 5 as the x-value.

Animation of a Graphical Image

Program *WALK* shows a small person walking across the display. It animates this custom graphical image by incrementing the image position in a loop structure.

Arguments	Results
`1:`	`1:`

Techniques

- Custom graphical image. (Note that the programmer compiles the full information content of the graphical image before writing the program by building the image *interactively* in the Graphics environment and then returning it to the command line.)
- FOR...STEP definite loop to animate the graphical image. The ending value for the loop is MAXR. Since the counter value cannot exceed MAXR, the loop executes indefinitely.

31

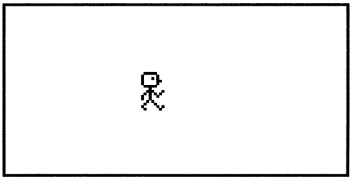
Program:	**Comments:**
`«`	
`GROB 9 15 E300 140015001C001400E300 8000C110AA0094009000 4100220014102800`	Puts the graphical image of the walker in the command line. (Note that the hexadecimal portion of the graphics object is a continuous integer `E300` ... `2800`. The linebreaks do *not* represent spaces.)
`→ walk`	Creates local variable *walk* containing the graphics object.
`«`	
`ERASE { # 0d # 0d } PVIEW`	Clears *PICT*, then displays it.

Program:	Comments:
`{ # 0d # 25d }` `PICT OVER walk GXOR`	Puts the first position on the stack and turns on the first image. This readies the stack and *PICT* for the loop.
`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.
`5 STEP`	Increments the horizontal coordinate by 5.
`»`	
`»`	
ENTER ' WALK STO	Stores the program in *WALK*.

Checksum: # 18146d
Bytes: 240.5

Example: Send the small person out for a walk.

VAR WALK



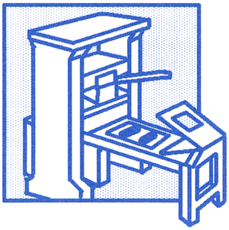
Press ATTN when you think the walker's tired.

Part 5

Printing, Data Transfer, and Plug-Ins

32

Printing



This chapter describes how to use your HP 48 with an HP 82240B infrared printer, with an HP 82240A infrared printer, and with printers that connect to the serial port.

Setting Up a Printer

You can print on any of the printers mentioned above. However, you must first make sure the HP 48 and the printer are set up properly.

32

To set up an HP 82240B printer:

1. Place the HP 48 and the printer on a flat surface. Aim the ▲ mark (near the Hewlett-Packard logo just above the display) toward the window on the printer. Keep them within 18 inches (45 centimeters).
2. Press 34 (+/-) (→)(MODES) (NXT) CF to make sure flag −34 is clear (its default state).
3. If you previously pressed OLDPR for any reason, purge variable *PRTPAR*—press (') (VAR) PRTPA (←)(PURGE).

To set up an HP 82240A printer:

1. Place the HP 48 and the printer on a flat surface. Aim the ▲ mark (near the Hewlett-Packard logo just above the display) toward the window on the printer. Keep them within 18 inches (45 centimeters).
2. Press 34 [+/-] [↱][MODES] [NXT] CF to make sure flag −34 is clear (its default state).
3. Press [↰][PRINT] [NXT] OLDPR to set up special processing for HP 48 characters.

The character set in the HP 82240A infrared printer doesn't match the HP 48 character set (see appendix C), but OLDPRT sets up the following adjustments:

- 24 characters in the HP 48 character set (codes 129, 130, 143–157, 159, 166, 169, 172, 174, 184, and 185) aren't available in the HP 82240A infrared printer—it prints ▒ instead.
- Many extended characters (codes 128 through 255) don't have the same character codes. For example, « has code 171 in the HP 48 and code 146 in the HP 82240A printer.

To cancel OLDPRT for an HP 82240A printer:

32

- Press ['] [VAR] PRTPA [↰][PURGE].

You need to cancel OLDPRT whenever you print a graphics object in graphics form.

To set up a serial printer:

- See "Setting Up a Serial Printer" on page 32-9.

See your printer manual for instructions about how to operate the printer.

Printing

With certain exceptions, printing commands print objects according to these guidelines:

- An object is printed with its delimiters.
- An object that doesn't fit in one line of output continues on the following lines.
- An array object is printed in expanded form—see below.
- A graphics object is printed in its stack form.

When you print an array in expanded form, each row and column is labeled. For example, the 2×3 array

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

is printed like this:

```
                 Array { 2 3 }   ← Array dimensions
Row number   →   Row 1
                 1] 1
Column number    2] 2
                 3] 3

                 Row 2
                 1] 4
                 2] 5
                 3] 6
```


You can perform any printing operation with any compatible printer—*with these exceptions*:

- Not all HP 48 characters print properly on an HP 82240A printer. (Press OLDPR in the PRINT menu to get as many correct characters as possible.)
- Special characters in the HP 48 character set may not print properly on a serial printer.
- You can't print a graphics object on a serial printer.

To print the object in level 1:

- Press [→] [PRINT] (*right*-shift).

  or
- Press [←] [PRINT] PR1.

PR1 prints a string with *no* delimiters. PR1 prints a graphics object in its *graphic* form. (OLDPRT must not be in effect while printing a graphics object.)

To print the display image:

- Hold down [ON], press and release [MTH], then release [ON].

  or
- Press [←] [PRINT] PRLCD.

This operation uses the current DELAY setting. To print the image to the serial port using [ON]-[MTH], first make sure the port is open—press OPENI in the I/O menu. (OLDPRT must not be in effect while printing a graphics object.)

Note A low-battery condition may result in consistent failure of the [ON]-[MTH] printing operation. If you notice consistent failure, replace your calculator batteries.

To print all objects on the stack:

- To print objects using multiple lines when necessary, press [←] [PRINT] PRST.
- To print objects truncated to one line each, press [←] [PRINT] PRSTC.

PRST and PRSTC print the stack starting with the object in the highest stack level.

To print objects stored in variables:

- To print one variable, enter its name (with ' delimiters) and press [←] [PRINT] PRVAR.
- To print several variables, enter a list (with { } delimiters) containing the variable names, then press [←] [PRINT] PRVAR.

PRVAR prints graphics objects in their *graphic* form. It also prints backup objects. PRVAR searches the current path for the variables you specify, and prints the name and contents of each variable. (OLDPRT must not be in effect while printing a graphics object.)

To print a string of characters:

1. Enter the characters as a string (with " " delimiters).
2. Press ⬅ PRINT PR1.

You can print any sequence of characters using PR1. The printer prints the characters without the " " delimiters. Subsequent printing begins on the next line.

To print a graphics object as a picture:

- To print the object in level 1, press ⬅ PRINT PR1.
- To print an object stored in a variable, enter its name (with ' delimiters) and press ⬅ PRINT PRVAR.
- To print a displayed object, press ON-MTH.

A graphics object wider than 166 dot columns is printed in 166-column wide segments down the paper, separated by a dashed line. For example, a 350-column wide graphics object would be printed in two 166-column segments and one 18-column segment. (OLDPRT must not be in effect while printing a graphics object.)

The following table summarizes the printing commands.

Printing Commands

Key	Programmable Command	Description
(→)(PRINT)	PR1	Prints the object in level 1.
(ON)-(MTH)		When (ON) and (MTH) are pressed simultaneously and then released, prints the current display.
(←)(PRINT):		
PR1	PR1	Prints the object in level 1.
PRST	PRST	Prints all objects on the stack, starting with the object in the highest level.
PRSTC	PRSTC	Prints all objects on the stack in compact form, starting with the object in the highest level.
PRLCD	PRLCD	Prints the current display.
PRVAR	PRVAR	Searches the current path for the specified variables, then prints the name and contents of each one. The variables are specified by a name or list in level 1.
CR	CR	Causes printer to do a carriage-return/line-feed, printing the contents, if any, of the printer buffer.
DELAY	DELAY	Sets the delay time (not greater than 6.9 seconds) between sending lines of information to the printer.
OLDPR	OLDPRT	Remaps character codes of printed output to those of the HP 82240A infrared printer.

Doing Advanced Printing

You can control other aspects of the printed output.

To set up double-spaced printing:

- To turn on double spacing, press 37 (+/-) (→)(MODES) (NXT) SF .
- To turn off double spacing, press 37 (+/-) (→)(MODES) (NXT) CF .

Flag −37, the Double-Spaced Printing flag, causes double-spaced printing when it's set.

To change the delay between printed lines:

- Enter the delay in seconds (not more than 6.9) and press (←)(PRINT) (NXT) DELAY .

The DELAY command lets you specify how long the HP 48 waits between sending lines of information to an infrared printer. The default delay is 1.8 seconds to avoid sending data faster than the printer can print.

You can use a shorter delay setting when the HP 48 sends multiple lines of information to your printer (for example, when printing a program). To optimize printing efficiency, set the delay just longer than the time the printhead requires to print one line of information.

32

If you set the delay *shorter* than the time to print one line, you may lose information. Also, as the batteries in the printer lose their charge, the printhead slows down, and, if you have previously decreased the delay, you may have to increase it to avoid losing information. (Battery discharge will not cause the printhead on an infrared printer to slow to more than the 1.8 second default delay setting.)

To include special HP 48 characters within a text string:

1. Enter in order *each part* of the string:
   - To enter normal text, enter it as a string (with " " delimiters) and press (ENTER).
   - To enter a special character, enter its character code and press CHR in the PRG OBJ menu.
2. Press (+) as needed to join the parts into the complete string.
3. Print the string using the PRINT menu.

The table in appendix C lists each HP 48 character and its corresponding *character code*. You can type most of the characters in the table from the Alpha keyboard—see the alpha keyboard diagram on page 2-8. For example, to type ‡, press [α] [◀] [4].

You can enter *any* HP 48 character using the CHR command. Certain characters in the table in appendix C are *not* on the Alpha keyboard. To enter one of these characters, you *must* use CHR.

The HP 82240B Infrared Printer can print any character from the HP 48 character set.

To include printer commands within a text string:

1. Build the text string—including the special printer command characters—using CHR and [+] as described above.
2. Print the string using the PRINT menu.

You can select various printer modes by sending *escape sequences* and *control characters* to the printer. (An escape sequence consists of the escape character—character 27—followed by additional characters.) When the printer receives an escape sequence or character code, it takes appropriate action—but the command itself isn't printed.

Printer owner's manuals usually describe the escape sequences and
32 control codes recognized by the printer.

Example: The following commands send information to the HP 82240B printer to turn on Underline mode, underline the string HELLO, and then turn off Underline mode:

```
27 CHR 251 CHR "HELLO" 27 CHR 250 CHR + + + + PR1
```

To accumulate data in the printer's buffer:

1. Press 38 [+/-] [▶] [MODES] [NXT] SF.
2. Use commands in the PRINT menu to send several batches of data to the printer.
3. Press CR in the PRINT menu to print the accumulated data.
4. Optional: Press 38 [+/-] [▶] [MODES] [NXT] CF to restore normal printing.

You can print any combination of text, graphics, and objects on a single print line by accumulating data in the printer's *buffer*. Normally, each print command completes data transmission by *automatically* executing the CR (*carriage right*) command, which tells the printer to do a carriage-return/line-feed. Then the printer prints the data currently in its buffer and leaves the print head at the right end of the print line. (Alternatively, send character 4 or character 10 to print the buffer.)

Flag −38, the Line-Feed flag, controls the automatic execution of the CR command. If it's set, data from subsequent print commands is accumulated in the printer buffer and is printed only when you manually execute CR.

For an infrared printer, follow these three rules while flag −38 is set:

- Execute CR before you accumulate more than 200 characters. Otherwise, the buffer fills up and subsequent characters are lost.
- Allow time for the printer to print a line before sending more data. The printer requires about 1.8 seconds per line.

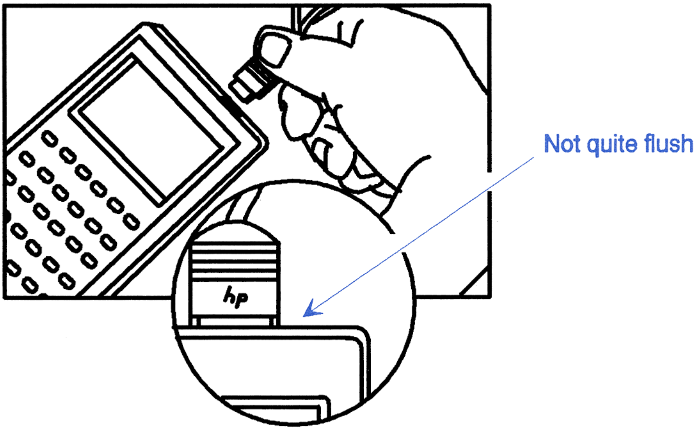
Setting Up a Serial Printer

You use the PC version of Serial Interface Cable to connect the HP 48 and the printer. This cable is also included with the Program Development Link, available from Hewlett-Packard. (For information about these and other products, see your HP dealer.) 32

To set up a serial printer:

1. Connect the 9-pin end of an HP 48 serial cable to the serial printer. If necessary, use a 9-pin to 25-pin adapter.

2. Keep the HP logo on the 4-pin connector facing up, then plug the cable into the HP 48. You should feel it lightly snap into place.



3. Press 34 [+/-] [↱][MODES] [NXT] SF to direct printing output to the serial port (instead of to the infrared port).

4. Press 33 [+/-] CF to make sure flag −33 is clear (its default state).

32

5. Press [↰][I/O] SETUP. Then check that the HP 48 and printer are using the same baud rate and parity. If desired, change the translate code to give more legible output. Other parameters don't affect printing. See "Setting the I/O Parameters" on page 33-3 for detailed information.

6. If your printer uses XON/XOFF handshaking: Press [↰][I/O] [NXT] OPENI CLOSE to create *IOPAR*. Then press [VAR] ['] IOPAR [↱][VISIT] and change the fourth number to 1—for example, { 9600 0 0 1 3 1 }. Press [ENTER].

7. If your printer doesn't fit 80 characters on one line, press [↰][PRINT] PR1 to create *PRTPAR*, then edit the *line-length* parameter—see "Understanding the PRTPAR Variable," the next topic.

8. If your printer requires an end-of-line sequence other than carriage-return/line-feed, press [↰][PRINT] PR1 to create

PRTPAR, then edit the *end-of-line* parameter—see "Understanding the PRTPAR Variable," the next topic.

If your printer doesn't use XON/XOFF handshaking, the printing *delay* parameter controls the time delay between lines—see the next section.

Understanding the PRTPAR Variable

When you first print information with a command from the PRINT menu, the HP 48 automatically creates the *PRTPAR* variable in the *HOME* directory. *PRTPAR* is a reserved variable containing a list that specifies how the HP 48 works with the printer:

{ *delay* "*remapping*" *line-length* "*end-of-line*" }

Contents of the PRTPAR List

Element	Description	Default
delay	A real number that specifies the delay time between sending lines, in seconds—6.9 maximum. (Set by DELAY in the PRINT menu.)	1.8
"*remapping*"	A string that represents the current remapping of the HP 48 extended character set. The string can contain as many characters as you want to remap. The first character in the string becomes the new character 128, the second becomes character 129, and so on. (Characters not represented in the string aren't remapped.) OLDPRT sets up the remapping string for the HP 82240A infrared printer.	""

32

Contents of the PRTPAR List (continued)

Element	Description	Default
line-length	A real number that specifies the line length, in number of characters, for serial printing only—it does *not* affect infrared printing.	80
"*end-of-line*"	A string that represents the line termination method for serial printing only—it does *not* affect infrared printing.	"■↵" (characters 13 and 10)

You can edit any parameter in variable *PRTPAR*. However, the remapping and end-of-line strings often contain special characters.

To edit the remapping or end-of-line string:

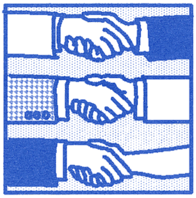
1. Create the desired string on the stack using CHR and (+). The procedure is explained on page 32-7.
2. Press (VAR) (') PRTPA (→)(VISIT)
3. Move the cursor to the start of the old string parameter and delete it—you can use DEL→.
4. Press ↑STK ECHO (ATTN) to insert the new string.
5. Press (ENTER) to save the change (or press (ATTN) to discard the change).

32

To reset PRTPAR to its default:

- Press (') (VAR) PRTPA (←)(PURGE).

Transferring Data to and from the HP 48



You can load data *into* your HP 48, and you can copy data *from* your HP 48. You can do this with two HP 48s—or with an HP 48 and a computer—or with an HP 48 and some other serial device, such as a printer.

How the HP 48 Transfers Data

The HP 48 uses *Kermit* file transfer protocol to transfer data and to correct transmission errors. Kermit protocol was developed at the Columbia University Center for Computing Activities and is implemented on most computers.

The commands needed to perform Kermit data transfers are built into the HP 48. You need nothing more to transfer data between two HP 48s.

To transfer data to and from a computer, the computer must be running a program that implements Kermit protocol. If you want additional information on Kermit protocol, the following books are available or can be ordered in many bookstores: *Using MS-DOS Kermit* by Christine M. Gianone, Digital Press, 1990, and *KERMIT, A File Transfer Protocol* by Frank da Cruz, Digital Press, 1987.

The HP 48 also provides commands for non-Kermit serial data transfers, such as sending data to a serial printer or instrument.

Types of Data You Can Transfer

The unit of information that you transfer using Kermit protocol is called a *file*. To the HP 48, a file can contain any of the following:

- A variable—any type of object stored there.
- An entire directory. When you transfer a directory, the contents of all the variables and subdirectories under that directory are also transferred.
- All of user memory—all the variables you've created, the user-key assignments, and the Alarm Catalog.

Whenever you transfer data, you actually send a *copy* of the data—*the original data is never removed*. The transferred data is stored in the current directory as a variable in another HP 48, or as a file on a computer.

When you transfer a directory from one HP 48 to another, it's created as a normal directory containing its individual variables. This means that you can use it just like other directories, and its variables are all accessible. Transferring a directory from one HP 48 to another is a good way to transfer a set of related objects to be used together by the destination HP 48. For example, it could contain a set of programs and related variables.

When you transfer a directory or all of user memory between an HP 48 and a computer, the data is embedded in a single file, so you can't conveniently access the contents of the individual variables in that file. For this reason, a directory transfer to a computer should be done mainly for archiving purposes. When the purpose of a file transfer is to use the file at its destination (for example, to edit a program on your computer), you should transfer the contents of the individual variables.

Choosing a Transfer Model

You can use two different Kermit protocol setups to transfer data between two devices:

- **Local/Local.** For each transfer, you operate *both* devices from their own keyboards—two "local" devices. (Kermit commands can be issued by both devices.)
- **Local/Server.** For each transfer, you operate *one* device—the "local" one. The other device—the "server"—takes its orders from the device you're operating. (Kermit commands can be issued by only the local device.)

Local/server mode is convenient because after you set up the server, you operate only one device.

Setting the I/O Parameters

The I/O parameters determine how the HP 48 communicates. In order for two devices to communicate, you have to make sure they use the same I/O parameters.

33

To view and change the HP 48 I/O parameters:

1. Press ⬅ I/O SETUP.
2. To change any parameter, press the corresponding menu key until the parameter has the desired value—see the table below.

Recommended parameters for certain types of transfers are summarized following this table.

Setup Commands

Key	Programmable Command	Description
⬅ I/O SETUP:		
IR/W		Switches between IR (infrared) and Wire (serial) modes. In IR mode, I/O output is directed to the infrared port. In Wire mode, I/O output goes to the serial port. (Flag −33 indicates this setting.)
ASCII		Switches between ASCII and Binary modes for sending data. (Flag −35 indicates this setting.)
BAUD	BAUD	Steps through 1200, 2400, 4800, and 9600 baud. (The default is 9600 baud.)
PARIT	PARITY	Steps through odd (1), even (2), mark (3), space (4), and no (0) parity. (The default is no parity.)
CKSM	CKSM	Steps through checksum (error detection) options—the type requested when initiating a SEND. Choices are 1 (one-digit arithmetic checksum), 2 (two-digit arithmetic checksum), and 3 (three-digit cyclic redundancy check, or CRC). Should be 3 for IR mode. (The default is 3.)
TRAN	TRANSIO	Steps through the character translation options (which character codes are translated): 0 (no translation), 1 (code 10) 2 (codes 128–159), or 3 (codes 128–255). Not used for binary transfers. (The default is 1.)
⬅ REVIEW		Redisplays the setup information.

33

The BAUD, PARITY, CKSM, and TRANSIO *commands* take a numeric argument from level 1.

Choosing ASCII or Binary Transfer

The HP 48 Kermit protocol provides two transfer modes—ASCII and Binary. To get the fastest transfers, you generally should use the following modes for *sending* data:

- For HP 48-to-HP 48 transfers, use Binary mode.
- For HP 48-to-computer transfers in which you'll view or edit the files on the computer, use ASCII mode.
- For HP 48-to-computer transfers in which you'll merely store the data on the computer, use Binary mode.

The HP 48 automatically uses Binary mode when sending libraries and backup objects, and when archiving all of user memory.

While *receiving data*, the HP 48 treats all files as ASCII unless they match the special encoding generated for HP 48 binary files—then the HP 48 automatically switches to Binary mode for files with such encoding.

In ASCII mode, characters are converted according to the character translation option. This makes it possible to view and edit such files on a computer. See "Understanding ASCII Transfers" on page 33-22. 33

In Binary mode, less processing is required. No character conversions are performed, so received files can't be displayed on a computer.

Choosing the Parity Option

If the parity setting is a positive number, it's used on both transmit and receive. If it's a negative number, it's used only on transmit, and parity isn't checked during receive. The menu key PARIT steps through only positive choices.

To set a negative (transmit-only) parity option, enter the option number and press [+/-] [→][ENTRY] PARIT [ENTER]. (You can also edit the parity parameter in *IOPAR*—see "Understanding the IOPAR Variable" on page 33-24.)

Choosing the Translation Option

The translation code affects only data transferred in ASCII mode. The preferred translation code depends on the type of transfer:

- For HP 48-to-HP 48 transfers, the translation code is ignored if you use Binary mode. (If you use ASCII mode, the translation code doesn't affect the results—just the speed.)
- For HP 48-to-computer and computer-to-HP 48 ASCII transfers, the translation code depends on the computer software you use to display or edit the file. See the discussion below. (For binary transfers, the translation code is ignored.)

The HP 48 character set contains certain characters that can't be displayed using most computer software packages. The characters with codes 160 through 255 require software that supports the ISO 8859 character set. The additional characters with codes 128 through 159 require software designed to support the HP 48, such as the Program Development Link from Hewlett-Packard.

The translate code determines what happens to these characters when they're sent or received by the HP 48:

- If you're using computer software that doesn't support some of the HP 48 characters, use translation code 2 or 3.
- If you're transferring strings containing binary data, use translation code 0.
- If you want to simply capture the ASCII data, use translation code 1 (the default).
- If the HP 48 is receiving data containing an `%%HP` header line, the translation code is ignored—see "Understanding ASCII Transfers" on page 33-22.

The next table shows the translations performed by the HP 48 during ASCII transfers. It shows the character codes and characters that are translated. The "10" and "10,13" entries indicate conversions between end-of-line sequences using line feed (10) and carriage return (13) characters. The *trans* entries are "backslash" translations that depict special HP 48 characters as ASCII text—they're defined in the second table. (Undefined "backslash" sequences are *not* changed.)

The second table shows the "backslash" translations for characters with code numbers above 127—they're used by translation codes 2 and 3 only, as shown in the first table.

Summary of ASCII Data Translation Options

Option 0	Option 1	Option 2	Option 3
Data Sent by HP 48			
	10 → 10,13	10 → 10,13 \ → \\ 128 → *trans* ⋮ 159 → *trans*	10 → 10,13 \ → \\ 128 → *trans* ⋮ 255 → *trans*
Data Received by HP 48			
	10,13 → 10	10,13 → 10 \\ → \ *trans* → *char* \000 → *char* ⋮ \159 → *char*	10,13 → 10 \\ → \ *trans* → *char* \000 → *char* ⋮ \255 → *char*

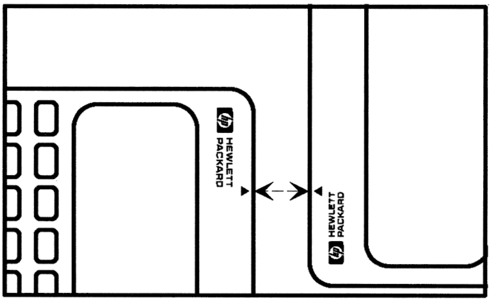
ASCII Character Translations (Character Codes 128–255)

<table>
<tr><th>HP 48 Code</th><th>HP 48 Char</th><th>Trans</th><th>HP 48 Code</th><th>HP 48 Char</th><th>Trans</th><th>HP 48 Code</th><th>HP 48 Char</th><th>Trans</th></tr>
<tr><td>128</td><td>∡</td><td>\<)</td><td>142</td><td>←</td><td>\<-</td><td>156</td><td>Π</td><td>\PI</td></tr>
<tr><td>129</td><td>x̄</td><td>\x-</td><td>143</td><td>↓</td><td>\|v</td><td>157</td><td>Ω</td><td>\GW</td></tr>
<tr><td>130</td><td>∇</td><td>\.V</td><td>144</td><td>↑</td><td>\|^</td><td>158</td><td>■</td><td>\[]</td></tr>
<tr><td>131</td><td>√</td><td>\v/</td><td>145</td><td>γ</td><td>\Gg</td><td>159</td><td>∞</td><td>\oo</td></tr>
<tr><td>132</td><td>∫</td><td>\.S</td><td>146</td><td>δ</td><td>\Gd</td><td>171</td><td>«</td><td>\<<</td></tr>
<tr><td>133</td><td>Σ</td><td>\GS</td><td>147</td><td>ε</td><td>\Ge</td><td>176</td><td>°</td><td>\^o</td></tr>
<tr><td>134</td><td>▶</td><td>\|></td><td>148</td><td>η</td><td>\Gn</td><td>181</td><td>µ</td><td>\Gm</td></tr>
<tr><td>135</td><td>π</td><td>\pi</td><td>149</td><td>θ</td><td>\Gh</td><td>187</td><td>»</td><td>\>></td></tr>
<tr><td>136</td><td>∂</td><td>\.d</td><td>150</td><td>λ</td><td>\Gl</td><td>215</td><td>×</td><td>\.x</td></tr>
<tr><td>137</td><td>≤</td><td>\<=</td><td>151</td><td>ρ</td><td>\Gr</td><td>216</td><td>Ø</td><td>\0/</td></tr>
<tr><td>138</td><td>≥</td><td>\>=</td><td>152</td><td>σ</td><td>\Gs</td><td>223</td><td>ß</td><td>\Gb</td></tr>
<tr><td>139</td><td>≠</td><td>\=/</td><td>153</td><td>τ</td><td>\Gt</td><td>247</td><td>÷</td><td>\:-</td></tr>
<tr><td>140</td><td>α</td><td>\Ga</td><td>154</td><td>ω</td><td>\Gw</td><td>nnn</td><td>other</td><td>\nnn</td></tr>
<tr><td>141</td><td>→</td><td>\-></td><td>155</td><td>Δ</td><td>\GD</td><td></td><td></td><td></td></tr>
</table>

Transferring Data between Two HP 48s

To set up for HP 48-to-HP 48 transfers:

1. **Sender.** Change to the directory where the variables to be sent are located.
2. **Sender.** Use the I/O SETUP menu to set up IR and Binary modes and checksum 3.
3. **Receiver.** Change to the directory where the variables are to be stored.
4. **Receiver.** Use the I/O SETUP menu to set up IR mode.
5. **Receiver.**
   - To allow received data to replace existing variables with the same names, set flag −36.
   - To resolve name conflicts by creating new names with number extensions, clear flag −36 (its default state).
6. Line up the infrared ports by lining up the ▲ marks (near the Hewlett-Packard logo just above the display). The calculators should be no farther apart than 2 inches.



To transfer a variable (local/local setup):

1. **Receiver.**
   - To store the variable using its original name, press (⬅)(I/O) RECV.
   - To store the variable using a new name, enter a name (with ' or " delimiters) and press (⬅)(I/O) (NXT) RECN.
2. **Sender.** Enter the name of the variable to be sent (with ' delimiters) and press (⬅)(I/O) SEND.

3. Optional: To transfer additional variables, repeat the previous steps.
4. **Sender and Receiver.** Optional: To conserve battery power, press (NXT) CLOSE.

To transfer a variable (local/server setup):

1. **Server.** Press (→)(I/O) *or* (←)(I/O) SERVE).
2. **Local.**
   - To send a variable, enter the variable name (with ' delimiters) and press (←)(I/O) SEND).
   - To get a variable, enter the variable name to get (with ' delimiters) and press (←)(I/O) KGET).
3. Optional: To get additional variables, repeat step 2.
4. **Local.** To end the session, press FINIS.
5. **Server and Local.** Optional: To conserve battery power, press (NXT) CLOSE.

To transfer several variables at once (either setup), use a list of the form { $name_1$ $name_2$ ... } as the argument for SEND or KGET.

To rename transferred variables (local/server setup), use a nested list of the form { { $name_{old}$ $name_{new}$ } ... } as the argument for SEND or KGET.

33

To send an HP 48 command to the server (local/server setup):

1. Enter the command as a string (with " " delimiters).
2. Enter the string "C".
3. Press (←)(I/O) (NXT) PKT .

Example: To purge variable *ABC* on the server, enter "'ABC' PURGE" and "C" and press PKT .

To end server mode on the HP 48:

- Press (ATTN).

Transferring Data between a Computer and HP 48

There are many reasons to transfer information between a computer and your HP 48—you might want to back up all of your calculator's user memory; you might want to edit a calculator program on your computer; or you might want to write a program on your computer and then run it on your calculator.

The Program Development Link from Hewlett-Packard provides a convenient way to transfer data between a computer and your HP 48. It's designed for creating and editing HP 48 programs. It automatically sets up I/O parameters for transferring data and backing up HP 48 memory.

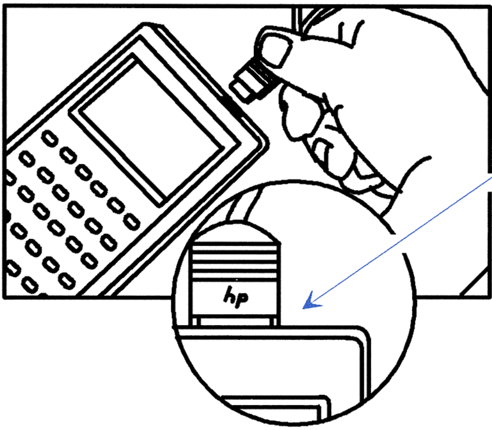
Preparing the Computer and HP 48

You use a Serial Interface Cable to connect the HP 48 and the computer. This cable is also included with the Program Development Link and with the Serial Interface Kit, available from Hewlett-Packard. (For information about these products, see your HP dealer.)

To connect a computer and HP 48:

1. Connect the computer end of the serial cable to the serial port on the computer. If necessary, use a connector adapter. (If you need more information, consult your computer documentation.)

2. Keep the HP logo on the 4-pin connector facing up, then plug the cable into the HP 48. You should feel it lightly snap into place.



To set up for HP 48-to-computer transfers:

1. **HP 48.** Press (←)(I/O) SETUP) and set up the following parameters:
   - Wire mode.
   - ASCII *or* Binary mode—it must match the Kermit mode on the computer. (See "Choosing ASCII or Binary Transfer" on page 33-5.)
   - Baud rate must match the Kermit baud rate on the computer.
   - Parity must match the Kermit parity on the computer.
   - Checksum can be any option—type 1 is the fastest.
   - Translation code can be any option. (See "Choosing the Translation Option" on page 33-6.)
2. **HP 48.** Change to the directory where the variables are to be sent from or stored.
3. **HP 48.**
   - To allow received data to replace existing variables with the same names, set flag −36.
   - To resolve name conflicts by creating new names with number extensions, clear flag −36 (its default state).
4. **HP 48.** Optional: Press (←)(I/O) (NXT) OPENI to open the HP 48 serial port. (This step isn't necessary for most connections, but it prevents difficulties caused by the inability of certain devices to communicate with a closed port.)

5. **Computer.** Change to the directory where the files are to be sent from or stored.
6. **Computer.** Run the program on the computer that implements Kermit protocol.
7. **Computer.** If you're using Binary mode, and if the Kermit program on the computer has a Binary mode command, execute the command.

Transferring Variables and Files

To send a file to the HP 48 (local/local setup):

1. **HP 48.**
   - To store the file in a variable of the same name, press ⬅ I/O RECV.
   - To store the file in a variable using a new name, enter a name (with ' or " delimiters) and press ⬅ I/O NXT RECN.
2. **Computer.** Execute the Kermit command to send the file, such as SEND *file*.
3. Optional: To transfer additional files, repeat steps 1 and 2.
4. **HP 48.** Optional: To conserve battery power, press NXT CLOSE.

To send a variable to the computer (local/local setup):

33

1. **Computer.** Execute the Kermit command to receive a file, such as RECEIVE.
2. **HP 48.** Enter the variable name (with ' delimiters) and press ⬅ I/O SEND.
3. Optional: To transfer additional variables, repeat steps 1 and 2.
4. **HP 48.** Optional: To conserve battery power, press NXT CLOSE.

To send several variables at once, use a list of the form { $name_1$ $name_2$ ... } as the argument for SEND.

To transfer data using the HP 48 (local/server setup):

1. **Computer.** Execute the Kermit command to make it the server, such as SERVER.
2. **HP 48.**
   - To send a variable, enter its name (with ' delimiters) and press ⬅ I/O SEND.
   - To receive a file into a variable, enter the file name (with " " delimiters) and press ⬅ I/O KGET.

3. Optional: To transfer additional variables, repeat step 2.
4. **HP 48.** To end the session, press FINIS.
5. **HP 48.** Optional: To conserve battery power, press NXT CLOSE.

To send several variables at once, use a list of the form { $name_1$ $name_2$ ... } as the argument for SEND.

To rename received variables, use a nested list of the form { { $name_{old}$ $name_{new}$ } ... } as the argument for KGET.

To transfer data using the computer (local/server setup):

1. **HP 48.** Press ↱ I/O *or* ↰ I/O SERVE).
2. **Computer.**
   - To send a file to the HP 48, execute the Kermit command to send the file, such as SEND *file*.
   - To receive a variable from the HP 48, execute the Kermit command to receive a file, such as RECEIVE.
3. Optional: To transfer additional variables, repeat step 2.
4. **Computer.** To end the session, execute the Kermit command to shut down the server, such as FINISH.
5. **HP 48.** Optional: To conserve battery power, press NXT CLOSE.

To send a command to a computer server (local/server setup):

1. **HP 48.** Enter the command as a string (with " " delimiters).
2. **HP 48.** Enter the string "C".
3. **HP 48.** Press ↰ I/O NXT PKT.

To send a command to an HP 48 server (local/server setup):

- **Computer.** Execute a Kermit REMOTE HOST *command* command, where *command* is one or more HP 48 commands or other objects.

To end server mode on the HP 48:

- **HP 48.** Press ATTN.

Backing Up All of HP 48 Memory

You can back up and restore the contents of the entire *HOME* directory in a file on your computer. The *HOME* directory includes all variables, user key assignments, and alarms. You can also include all flag settings if you want.

The Program Development Link from Hewlett-Packard provides commands for automatically backing up and restoring HP 48 memory from a computer.

The following steps assume you've prepared the computer and HP 48 for data transfer—see "Preparing the Computer and HP 48" on page 33-10.

Caution While backing up memory, make sure the ticking clock is not in the display. If the clock is in the display, it may cause a loss of data stored in memory after the backup is complete.

To back up all of user memory to a computer file:

1. **Computer.** Execute the Kermit command to set up binary transfer, if available.
2. **Computer.** Execute the Kermit command to receive a file or make it the server, such as RECEIVE or SERVER.
3. **HP 48.** Optional: To back up flags settings too, press [→][MODES] [NXT] RCLF , enter a flag-variable name (with ' delimiters), and press [STO].
4. **HP 48.** Enter the tagged object `:IO:`*name* on the stack, where *name* is the name of the file to be created on the computer.
5. **HP 48.** Press [←][MEMORY] [NXT] [NXT] ARCHI .
6. **HP 48.** To end the session, press [←][I/O] FINIS .
7. **HP 48.** Optional: To conserve battery power, press [NXT] CLOSE .

ARCHIVE always uses binary transfer, regardless of the ASCII/Binary setting on the HP 48.

Caution

Use the RESTORE command with care; restoring backed up user memory completely erases current user memory and replaces it with the backup copy.

To restore HP 48 user memory from a computer file:

1. Transfer the computer file to an HP 48 variable using one of the data transfer methods from the previous section.
2. **HP 48.** Enter the received variable name (with ' delimiters) and press (→)(RCL) to recall the backup object.
3. **HP 48.** Press (←)(MEMORY) (NXT) (NXT) RESTO.
4. **HP 48.** Optional: To restore flag settings previously saved, enter the flag-variable name (with ' delimiters), press (→)(RCL), and press (→)(MODES) (NXT) STOF.
5. **HP 48.** Optional: To conserve battery power, press (←)(I/O) (NXT) CLOSE.

Example: To back up memory into a file named *AUG1*, enter the tagged object :IO:AUG1 as the backup name. Then, if you later retrieve this data to the HP 48, you can enter 'AUG1' and press (→)(RCL) to get Backup HOMEDIR on the stack—ready for the RESTORE command.

33

Choosing and Using File Names

The naming conventions for computer files are different from those for HP 48 variables.

When the HP 48 *receives* a file from a computer, certain difficulties may arise due to the computer file name. (You can avoid this problem by specifying a new name for received data, as described in the transfer instructions.)

- If the file name contains characters not allowed in a variable name (such as AB# or {ABC}), the HP 48 terminates the transfer and sends an error message to the computer.

- If the file name matches a built-in command (such as SIN or DUP), the HP 48 appends a number extension to the name (such as SIN.1).
- If the name matches a variable name in the current directory and flag −36 is clear (to protect existing variables), a number extension is added to the name (such as NAME.1).

When the HP 48 *sends* a variable to a computer, its name may be incompatible with the naming conventions of the computer software. Transferring such a file can result in a transfer error. (You can avoid this problem by renaming the variable before sending it.)

Receiving Data from Other Calculators

The HP 48 is capable of receiving data from another calculator that has an infrared printer output—data is received as string objects. To do this, you need the INPRT program for the HP 48, which is available in the Serial Interface Kit and electronically on the HP Calculator Bulletin Board system—see the inside back cover.

33

Sending Kermit Commands

If the HP 48 is the *local* device in a local/server setup, you can use it to send Kermit commands to be executed by the server—by another HP 48 or by a computer. If the HP 48 is a *server*, you can send Kermit commands to it. The following steps assume the receiving device is already set up as a server.

To send a Kermit command from an HP 48:

1. Enter the command as a string (with " " delimiters).
2. Enter the packet type as a string (with " " delimiters).
3. Press (⬅)(I/O) (NXT) PKT .

The server sends one of the following responses to the PKT command:

- An acknowledging message. The reply to the packet is returned as a string to level 1—an empty string is returned if no response is appropriate.
- An error packet. The HP 48 briefly displays the contents of the error packet. To retrieve it, press ⬅ I/O NXT KERR.

Example: To request a directory listing, enter "DIRECTORY" and "G" and press PKT. The directory is returned as a string.

Getting Information about Kermit Errors

If a Kermit error occurs during a transfer, the transfer has failed. In this situation, you'll usually see a message in the HP 48 display, such as Invalid Syntax and additional information.

To recall the complete Kermit error packet:

- Press ⬅ I/O NXT KERR.

The KERRM command returns the most recent Kermit error packet as a string. The string is cleared by the CLOSEIO command. 33

Summary of Kermit Commands

The I/O Menu—Kermit Commands

Key	Programmable Command	Description
⬅ I/O (pages 1 and 2):		
SEND	SEND	Sends the contents of one or more variables to another device. Takes an argument from level 1—the variable name or a list of names.
RECV	RECV	Tells the HP 48 to wait to receive a variable from another Kermit device.

The I/O Menu—Kermit Commands (continued)

Key	Programmable Command	Description
SERVE	SERVER	Puts the HP 48 into Kermit Server mode. (Also ⮫ I/O.) Press ATTN to cancel.
KGET	KGET	Gets one or more variables from a server device. Takes an argument from level 1—the name of the requested variable or a list of names.
FINIS	FINISH	Issues the Kermit FINISH command to a server device to terminate Server mode.
SETUP		Displays the SETUP menu for setting I/O parameters.
RECN	RECN	Same as RECV for one variable, except that it takes a name argument. The received file is stored using that name.
PKT	PKT	Provides the ability to send a Kermit command "packet" to a server. It takes the packet data field as a string in level 2 and the packet type as a string in level 1.
KERR	KERRM	Returns the text of the most recent Kermit error.
OPENI	OPENIO	Opens the serial port using the I/O parameters in *IOPAR*.
CLOSE	CLOSEIO	Closes the serial port, clears the KERRM error message, and clears the input buffer.
⮫ I/O		Puts the HP 48 into Kermit Server mode. (Same as SERVE.) Press ATTN to cancel.

33

Sending and Receiving Data without Kermit

You can send and receive data and commands with serial devices that *don't* use Kermit protocol, such as serial printers and instruments. You do this using the general-purpose serial I/O commands.

To transfer serial data with a non-Kermit serial device:

1. Press ⬅ I/O `SETUP` and set up the I/O parameters to match the serial device.
2. If the serial device uses receive or transmit pacing (XON/XOFF signals) during transfers, press ⬅ I/O NXT `OPENI` `CLOSE` to make sure *IOPAR* exists, then press VAR ' `IOPAR` ➡ VISIT:
   - To receive data using pacing, change the third number to 1.
   - To send data using pacing, change the fourth number to 1—for example, `{ 9600 0 0 1 3 1 }`.

   Press ENTER.
3. Optional: Press ⬅ I/O NXT `OPENI` to open the HP 48 serial port. (This step isn't necessary for most connections, but it prevents difficulties caused by the inability of certain devices to communicate with a closed port.)
4. To send or receive serial data or commands, use the I/O menu keys for the desired operations—see the table below.

Caution

When using the commands described below to transfer data at 9600 baud, make sure the ticking clock is not in the display. If the clock is in the display, it may interrupt a transfer or corrupt the data being transferred.

The I/O Menu—Serial I/O Commands

Key	Programmable Command	Description
⬅ I/O (page 3):		
XMIT	XMIT	Sends the string in level 1 without Kermit protocol. After the entire string is sent, 1 is returned to level 1. If the entire string failed to transmit, 0 is returned to level 1 and the unsent part of the input string is returned to level 2—execute ERRM to see the error message.
SRECV	SRECV	Receives the number of characters specified in level 1. For a successful transfer, the characters are returned to level 2 as a string, and 1 is returned to level 1. For an unsuccessful transfer, an empty or incomplete string is returned to level 2, and 0 is returned to level 1—execute ERRM to return the error message. (An unsuccessful transfer occurs if the characters contain a parity error, framing error, or overrun error, or if fewer than the specified number of characters are received before the timeout period expires, 10 seconds by default.) Characters are taken from the input buffer—no waiting occurs if you specify the number of characters in the buffer, which is returned by BUFLE.
STIME	STIME	Sets the serial transmit/receive timeout to the number of seconds specified in level 1. The timeout value can be from 0 to 25.4 seconds. If you specify 0, the HP 48 waits indefinitely, which could result in excessive battery drain.
SBRK	SBRK	Sends a serial BREAK signal.

33

The I/O Menu—Serial I/O Commands (continued)

Key	Programmable Command	Description
BUFLE	BUFLEN	Returns the *number* of characters in the input buffer to level 2, and the error status to level 1 (1=no framing error or UART overrun, or 0=framing error or UART overrun). If 0 is returned to level 1, the number of characters returned to level 2 represents the part of the data received *before* the error—you can use it to determine where the error occurred.

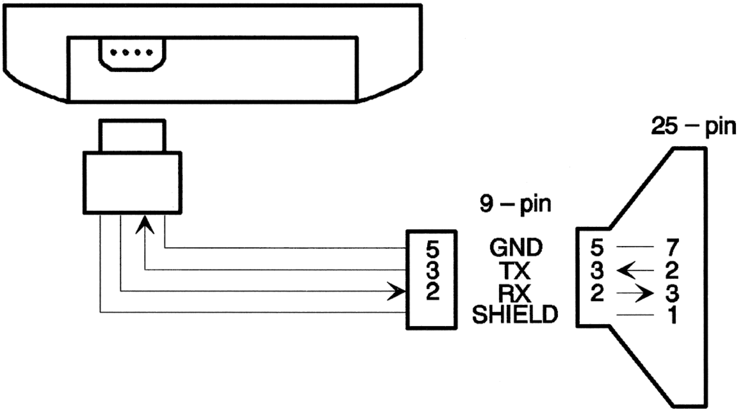
Note

Although XMIT, SRECV, and BUFLEN check the send and receive mechanisms, the integrity of the data isn't checked. One method to check the integrity of data transmission is for the sending device to append a checksum to the end of the data being sent, and for the receiving device to verify the checksum.

OPENIO, XMIT, SRECV, and SBRK automatically open the IR/serial port using the current values of the first four *IOPAR* parameters (baud, parity, receive pacing, and transmit pacing) and the current IR/wire setting (set using IR/W in the I/O SETUP menu). If you open the port, the input buffer can receive incoming data (up to 255 characters), even before you execute SRECV.

Making a Serial Connection

You normally use a Serial Interface Cable to connect the serial port. This cable is also included with the Program Development Link and with the Serial Interface Kit, available from Hewlett-Packard. (For information about these products, see your HP dealer.) The following diagram shows the wiring used by the PC version of the serial cable and its adapter.



33 **Serial Interface Cable and Adapter (PC Version)**

Understanding ASCII Transfers

You *must* use ASCII mode if you want to display, edit, or print your HP 48 file using a computer.

When data is *sent* from the HP 48 in ASCII mode:

- The data is converted from its internal HP 48 format to a sequence of characters.
- An %%HP header line is added at the beginning of the data. It describes certain current settings—the translation code, angle mode, and fraction mark.

When data is *received* by the HP 48 in ASCII mode:

- The data is translated (compiled) into the HP 48 internal format.
- If an `%%HP` header line is present, all modes specified in the line are set temporarily in the HP 48 for the duration of the transfer—so that the receiving calculator can accurately reconstruct the object being sent by the computer. If a mode isn't specified—or if no header line is included—the HP 48 uses its current setting.

The `%%HP` header line provides a convenient way to set up the translation code, angle mode, and fraction mark—without having to check their settings at the time of transfer. This is the format of the header line:

`%%HP:T(`*code*`)A(`*angle*`)F(`*mark*`);`

where

`T(`*code*`)` If present, specifies the translation code used when the HP 48 *receives* the data: `T(0)` (no translation), `T(1)` (code 10), `T(2)` (codes 128–159), or `T(3)` (codes 128–255). See "Choosing the Translation Option" on page 33-6.

`A(`*angle*`)` If present, specifies the angle mode used when the HP 48 *receives* the data: `A(D)` (degrees), `A(R)` (radians), or `A(G)` (grads). If the data contains an angle, this setting is important.

`F(`*mark*`)` If present, specifies the fraction mark used when the HP 48 *receives* the data: `F(.)` (period) or `F(,)` (comma). If the data contains a fraction mark, this setting is important.

If you use your computer to create data (such as an HP 48 program) or to substantially change data that originally came from your calculator, you may want to include an `%%HP` header line at the beginning of the file. It ensures that the file is transferred and interpreted correctly.

Example: The header line `%%HP:A(D);` causes the angle mode to be set to degrees during the transfer—the current translation code and fraction mark are used.

Example: The header line `%%HP:T(2)A(G)F(,);` causes the translate code to be set to 2, the angle mode to be set to grads, and the fraction mark to be set to comma during the transfer.

Understanding the IOPAR Variable

The reserved variable *IOPAR* stores the I/O parameters needed to establish a communications link with a computer. It's created in the *HOME* directory the first time you transfer data or open the serial port (OPENI). It's automatically updated whenever you change the settings using the commands in the I/O SETUP menu. *IOPAR* contains a list consisting of these elements, described in the table following:

{ *baud* *parity*
receive-pacing *transmit-pacing* *checksum* *translation-code* }

Contents of the IOPAR List

Element	Description	Default
baud	Baud rate (1200, 2400, 4800, or 9600).	9600
parity	Parity (0=none, 1=odd, 2=even, 3=mark, 4=space).	0
receive-pacing	For non-Kermit transfers only (0=disabled, *nonzero*=enabled). Receive pacing sends XOFF when the receive buffer is almost full, and sends XON when it can take more data.	0
transmit-pacing	For non-Kermit transfers only (0=disabled, *nonzero*=enabled). Transmit pacing stops transmission when XOFF is received, and resumes when XON is received.	0
checksum	Error-detection scheme for SEND (1=one-digit checksum, 2=two-digit checksum, 3=three-digit cyclic redundancy check).	3
translation-code	Character translations (0=none, 1=code 10, 2=codes 128–159, 3=codes 128–255). Not used for binary transfers. See "Choosing the Translation Option" on page 33-6.	1

34

Memory, Plug-In Cards, and Libraries

The HP 48 contains built-in permanent memory (ROM) and built-in user memory (RAM). A special built-in memory port (port 0) is also available for memory operations. *Except for the HP 48S model*, the HP 48 also has two plug-in ports (ports 1 and 2) that let you add to built-in memory by plugging in application cards and RAM cards.

This chapter shows:

- The types of memory.
- Installing and removing plug-in cards (not for the HP 48S).
- Expand user memory (not for the HP 48S).
- Backing up data.
- Using libraries.

34

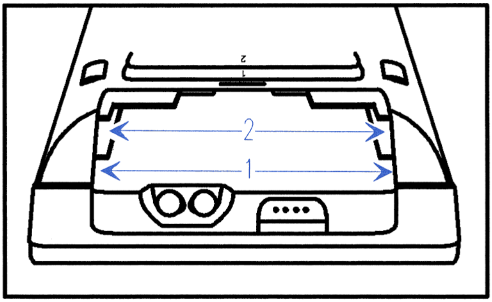
Types of Memory

The HP 48 has two types of memory:

- **Read-only memory (ROM).** Memory that can't be altered. The HP 48 has 256 KB of built-in ROM that contains its command set. Except for the HP 48S, you can expand the amount of ROM by installing plug-in application cards.
- **Random-access memory (RAM).** Memory you can change. You can store data into RAM, modify its contents, and purge data. The HP 48 contains 32 KB of built-in RAM. Except for the HP 48S, you can increase the amount of RAM by adding plug-in RAM cards.

Installing and Removing Plug-In Cards (Not HP 48S)

The two ports for installing plug-in cards are designated port 1 and port 2. Port 1 is closest to the front of the calculator—port 2 is closest to the back. You can install a card in either port.



If you're installing a *new* RAM card, read the next section—otherwise, skip ahead to "Installing and Removing RAM and ROM Cards" on page 34-5.

34

Caution

Nonapproved plug-in cards and accessories may cause damage to the HP 48. You can distinguish a potentially damaging card or plug-in accessory from an HP-approved card by looking at the back side of the card where it plugs into the HP 48. An approved card has a metal shutter to protect the HP 48 from static charges. The nonapproved cards and accessories examined to date by HP do not have this shutter, but have exposed gold contacts instead.

Preparing a New RAM Card

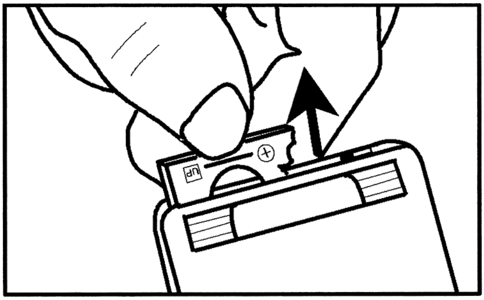
Before you install a new RAM card, you must install the battery that came with it.

Caution

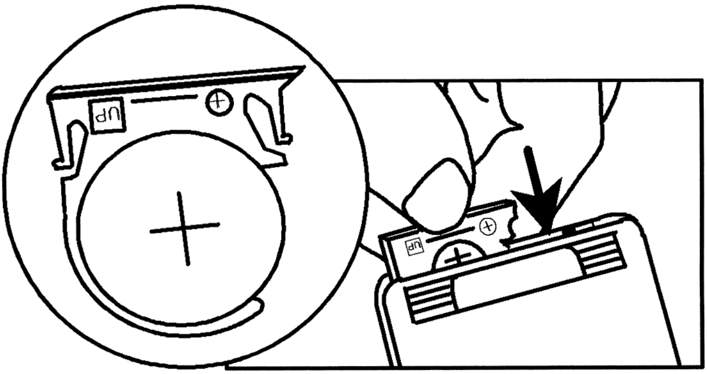
Do *not* use this procedure for *replacing* the battery in a RAM card—it could cause loss of memory in the RAM card. To *replace* a battery, see "To change a RAM card battery" on page A-9.

To install the battery in a new RAM card:

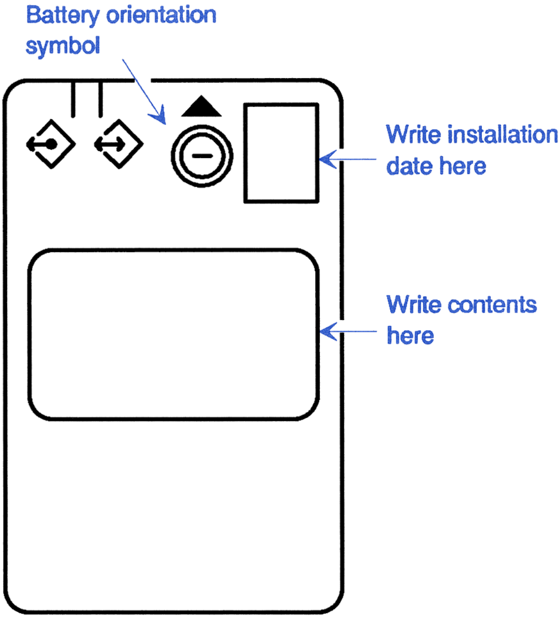
1. Remove the battery holder from the card by inserting your thumbnail or a small screwdriver into the groove and pulling in the direction shown.



2. The grooved side of the battery holder is marked with the + symbol and the word UP. Insert the battery into the holder with its + side up, and then slide the holder into the card.



3. Write the date of installation on the card using a fine-point, permanent marker. The date is important for determining when to replace the battery.



34

4. Set an alarm in the calculator for 1 year from the date of installation to remind you to replace the battery. (Depending on the use, the battery should last between 1 and 3 years. When the battery needs replacing, a display message will appear—*but only if the card is in the calculator.* You set this alarm to remind yourself in case the card isn't in the calculator when the battery gets low.) To set an alarm, see "Setting Alarms" on page 24-5. To replace a RAM-card battery, see "To change a RAM card battery" on page A-9.

Installing and Removing RAM and ROM Cards

Caution

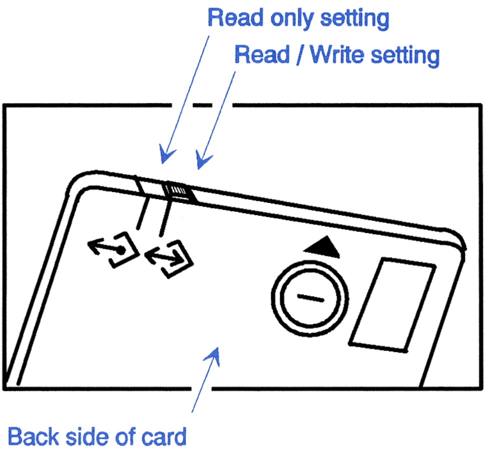
Turn off the calculator *before* you install or remove a plug-in card. Otherwise, all of user memory could be erased.

Also, whenever you install or remove a card, the HP 48 executes a system halt, causing the contents of the stack to be lost. See "Saving and Restoring the Stack" on page 5-3.

To install a plug-in card:

1. If you're installing a *new* RAM card, first install its battery—see the previous section.
2. For a RAM card, check or set the write-protect switch. *For a new RAM card, set it to Read/Write.*
   - **Read Only.** You can read the contents of the card, but you can't change, erase, or store data. It protects the contents of the RAM card from being accidentally overwritten or erased.
   - **Read/Write.** You can read, change, and erase the contents and store data, as you do with built-in user memory.

34

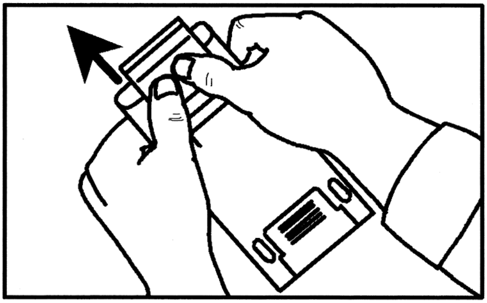


Caution

To avoid loss of user memory:

- Always turn off the calculator before changing the write-protect switch on an installed card.
- Do not write protect a RAM card containing merged memory—you should write protect only *independent* memory.

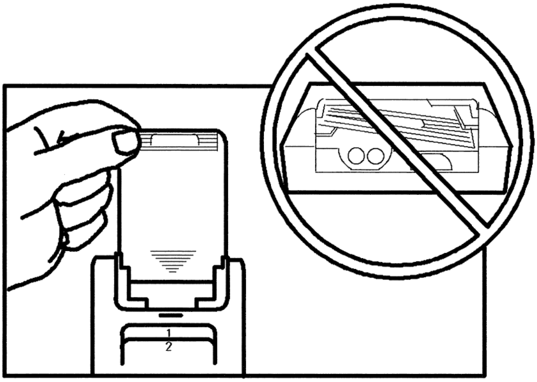
3. Turn off the calculator. *Do not press* [ON] *until you've completed the installation procedures.*
4. Remove the port cover at the top of the calculator by pressing down against the grip area and then pushing in the direction shown. Removing the cover exposes the two plug-in ports.



5. Select an empty port for the card—you can use either port.

34

6. Position the plug-in card as shown. The triangular arrow on the card must point down, toward the calculator. Make sure the card is lined up properly with a port opening and not positioned half in one port and half in the other.



7. Slide the card firmly into the port until it stops. When you first feel resistance, the card has about 1/4 inch to go to be fully seated.
8. Replace the port cover by sliding it on until the latch engages.
9. Press (ON) to turn on the calculator.

34

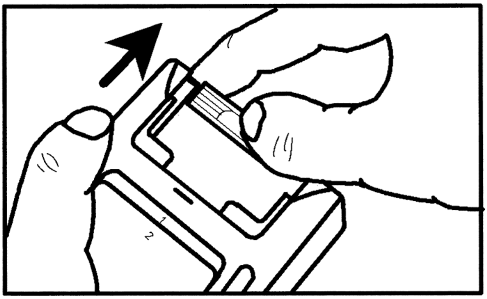
Note

When you install a new RAM card and turn on the calculator, you get the message `Invalid Card Data` because the card isn't initialized. Disregard the message—the card is automatically initialized the first time you use it.

To remove a plug-in card:

1. *If you're removing a RAM card, make sure it contains independent memory*—see the caution below and "Merging, Freeing, and Protecting Memory" on page 34-11.
2. Turn off the calculator. *Do not press* (ON) *until you've removed the card.*
3. Remove the port cover.

4. Press against the grip as shown and slide the card out of the port.



5. Replace the port cover.

Caution

Never remove a RAM card that contains *merged* memory—it will probably cause a loss of data stored in user memory. Before you remove the RAM card, you must *free* the merged memory. See "Merging, Freeing, and Protecting Memory" on page 34-11.

If you accidentally remove a card with merged memory and see the message `Replace RAM, Press ON`, you can minimize memory loss by *leaving the calculator on*, reinserting the card in the *same* port, and then pressing [ON].

34

Using Plug-In Cards (Not HP 48S)

You can extend built-in HP 48 memory by installing a plug-in RAM card or application card in port 1 or port 2. (The HP 48S has no plug-in ports.)

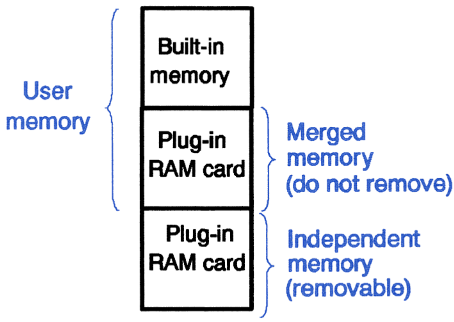
Using RAM Cards

RAM cards let you increase the amount of RAM in your HP 48. Each RAM card contains a battery that preserves its contents while the calculator is turned off and after you've properly removed the card from the calculator. (The calculator batteries power the RAM card only while the calculator is turned on.)

You set up a RAM card as one of two types of memory—each with its own benefits. You can change between the two types—but you can't use one card as both types at the same time. If you install *two* RAM cards, you can set up each card individually.

- **Merged memory.** The part of user memory that's contained in a RAM card—the card's memory is *merged* with built-in user memory. This lets you to expand the amount of user memory for creating variables and directories, and for putting objects on the stack. See "Expanding User Memory" on page 34-14.
- **Independent memory.** RAM memory that's *independent* of user memory—in built-in memory (in port 0) or in a RAM card (in port 1 or 2). This lets you back up individual objects or entire directories, much as you'd back up computer files to a disk, then store it in a safe place. You can also use it to transfer data to another HP 48 by installing it and copying the objects there. See "Backing Up Data" on page 34-15.

The following diagram illustrates a system containing two RAM cards—one containing merged memory and the other containing independent memory.

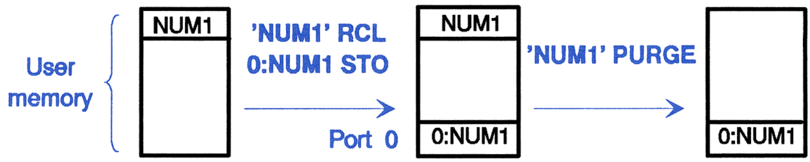


Using Application Cards

Application cards typically contain library objects, which can act as extensions to the built-in command set. See "Using Library Objects" on page 34-19.

Using Port 0

Port 0 is a special part of memory that operates as independent memory, similar to independent memory in RAM cards in plug-in ports. However, port 0 is available even if there are no plug-in ports, such as in the HP 48S. The memory for port 0 is taken out of user memory—so objects stored in port 0 decrease the amount of user memory available. The size of port 0 is dynamic—it grows and shrinks to accommodate its contents.



If you don't have or don't want to use port 1 or 2, you can use port 0 for storing backup objects and library objects. You can also use port 0 to "hide" data—that is, to have certain variables available in memory but not appear in any directory.

Merging, Freeing, and Protecting Memory (Not HP 48S)

When you first install a RAM card, it's set up as *independent* ("free") memory—you can use it to store backup objects and libraries. To protect your backup objects and libraries, you can use the card's write-protect switch to prevent altering the data.

If you want to expand user memory, you set up the card as *merged* memory. You must not write-protect a merged card—the HP 48 must be able to access user memory at all times.

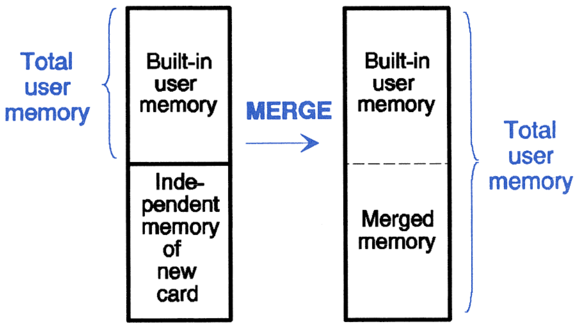
To check the type of memory in a port:

- Enter the port number (1 or 2) and press [←] [MEMORY] [NXT] PVARS. The result in level 1 indicates the type of memory:

"ROM"	ROM in an application card.
"SYSRAM"	Merged memory in a RAM card.
number	Independent memory in a RAM card.

To use an installed RAM card to expand user memory:

1. Make sure the card is *not write-protected*—make sure its switch is away from the corner of the card. (*Turn off the calculator if you need to change it.*)
2. Enter the port number that the card is installed in (1 or 2) and press [←] [MEMORY] [NXT] [NXT] MERG.



34

If the card previously contained any backup objects or libraries, the MERGE command automatically moves them to a special part of memory called port 0. See "Using Port 0" on page 34-10.

Caution

Never remove a RAM card that contains *merged* memory—it will probably cause a loss of data stored in user memory. Before you remove the RAM card, you must *free* the merged memory—see the steps below.

If you accidentally remove a card with merged memory and see the message `Replace RAM, Press ON`, you can minimize memory loss by *leaving the calculator on*, reinserting the card in the same port, and then pressing [ON].

To free a card that's merged into user memory:

1. Press [←] [{ }] [ENTER] to enter an empty list.
2. Enter the port number that the card is installed in (1 or 2).
3. Press [←] [MEMORY] [NXT] [NXT] `FREE`. (If you get an error, see below.)
4. Optional: Turn off the HP 48 and unplug the card—see "To remove a plug-in card" on page 34-7.

If the RAM card is already free (independent memory), you'll get a `Port Not Available` error when you execute FREE.

If there isn't enough memory available to free the RAM card, you'll get a memory error when you execute FREE. To check for this condition, press [←] [MEMORY] `MEM` —the number returned is the amount of unused user memory in bytes. *To be able to free the RAM card, you must have an unused amount that's greater than or equal to the size of the RAM card*—otherwise, the HP 48 doesn't have enough unused memory to allocate to the card.

If you don't have enough available memory to free the RAM card, you can try these ideas to make the available memory large enough:

- Purge unneeded variables from user memory.
- Back up data into another RAM card installed in the other port and then purge the original variables.

- Back up data into port 0, then move the backup objects to the RAM card as it's being freed:
  1. Determine the amount of data you need to remove from user memory. (For example, if you're removing a 128-KB RAM card and the amount of unused user memory is 126 KB, you must move at least 2 KB of variables.)
  2. Back up that amount of data into port 0 and delete the original variables.
  3. Free the card *and* move the backup objects there—see the next steps below.

To free a merged RAM card and move backup objects there:

1. Back up the desired objects into port 0—see "To back up an object" on page 34-15.
2. Enter a list (with { } delimiters) containing the simple names of the backup objects in port 0.
3. Enter the port number that the card is installed in (1 or 2).
4. Press (⬅)(MEMORY) (NXT) (NXT) FREE .
5. Optional: Turn off the HP 48 and unplug the card—see "To remove a plug-in card" on page 34-7.

The objects named in the list are removed from port 0 and stored in the newly freed RAM card (in independent memory).

34

Example: Assume backup objects *NUM1* and *ADD3* are stored in port 0, and a RAM card in port 1 is set up as merged user memory. To move these two objects to the RAM card so you can store them for safekeeping, enter { NUM1 ADD3 }, enter 1, and execute FREE. Now you can remove the card—it contains the two backup objects, which were also deleted from port 0.

Caution

To avoid loss of user memory:

- Always turn off the calculator before changing the write-protect switch on an installed card.
- Do not write protect a RAM card containing *merged* memory—you should write protect only *independent* memory.

To change the write-protect switch with the card installed:

1. *Make sure the card contains independent memory*—see "To check the type of memory in a port" on page 34-11.
2. Turn off the HP 48.
3. Move the switch to the correct position:
   - For Read Only, the switch is *toward* the corner of the card.
   - For Read/Write, the switch is *away from* the corner of the card.

Expanding User Memory (Not HP 48S)

You can use a RAM card to expand user memory—just set up the card as merged memory. If you want to remove the card later, first you have to free the merged memory.

Caution

To avoid loss of user memory:

- Do not write protect a RAM card containing *merged* memory.
- Do not unplug a RAM card containing *merged* memory.

To set up an installed RAM card to expand user memory:

- Merge the RAM card memory—see "Merging, Freeing, and Protecting Memory" on page 34-11.

To free or remove a card that's merged into user memory:

- Free the RAM card memory—see "Merging, Freeing, and Protecting Memory" on page 34-11.

Backing Up Data

The HP 48 uses a special object type, the *backup object*, to store backup data. A backup object contains another object, its name, and its checksum. Simply put, a backup object contains a variable or directory and its checksum.

Backup objects can exist only in *independent memory*:

- Port 0.
- Ports 1 and 2 if they contain RAM cards set up as independent memory. When you first install a card, it's set up as independent memory. (Ports 1 and 2 don't exist in the HP 48S.)

To set up a card as independent memory:

- Free the RAM card memory—see "Merging, Freeing, and Protecting Memory" on page 34-11. (If you haven't set up the card as *merged* memory, then it's already set up as *independent* memory.)

Backing Up Individual Objects

To back up an object:

34

1. Put the object on the stack.
2. Enter a *backup identifier* for the backup object to create—see below.
3. Press STO.
4. Optional: Purge the original object in user memory.

The STO command creates the backup copy using the port and name specified by the *backup identifier*—it has the form

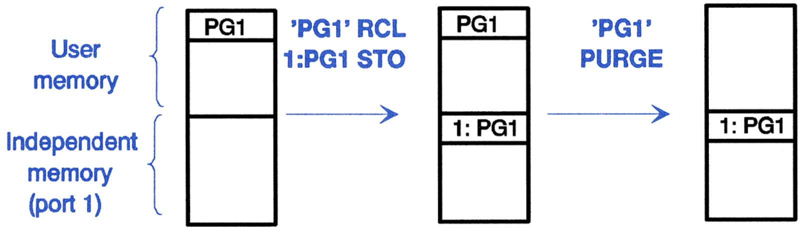
:*port*: *name*

where *port* is the port number (0, 1, or 2), and *name* is the name where the backup copy is stored. If you use port 1 or 2, it must be set up as independent memory. The name of the backup object can be different from the original name.

You can back up an entire directory (and its subdirectories) in one backup object by putting the directory object on the stack and making a backup copy.

If a backup object exists with a given backup identifier, you have to purge the backup object before you can use the identifier for another backup object.

Example: To back up a program named *PG1* into independent memory in port 1, recall the program to the stack by entering `'PG1'` and pressing [→] [RCL], then enter `:1:PG1` and press [STO].



Example: To back up the subdirectory named *CHEM* (in the *HOME* directory) in a backup object named *BCHEM*, press [→] [HOME] [VAR] [→] `CHEM` to put the directory on the stack, then enter `:1:BCHEM` and press [STO].

To display a port menu of backup objects and libraries:

1. Press [←] [LIBRARY].
2. Press `PORT0`, `PORT1`, or `PORT2` for the port you want.

`PORT0`, `PORT1`, or `PORT2` displays a menu of backup objects and libraries in that port.

To enter the backup identifier of a backup object:

- Display the appropriate PORT menu, then press [→] [ENTRY], the menu key for the object, and [ENTER].

To recall a backup object to the stack:

- Display the appropriate PORT menu, then press [→] and the menu key for the object.

  or
- Enter the backup identifier for the backup object and press [→] [RCL].

To evaluate a backup object:

- Display the appropriate PORT menu, then press the menu key for the object.

  or
- Enter the backup identifier for the backup object and press (EVAL).

To evaluate several backup objects in a row, enter a list (with { } delimiters) containing the backup identifiers, then press (EVAL).

Example: To run the backup-object program stored in port 0 with the name *BPRG*, press (←)(LIBRARY) PORT0 BPRG .

To delete a backup object:

- Enter the backup identifier for the backup object and press (←)(PURGE).

To purge several backup objects, enter a list (with { } delimiters) containing the backup identifiers, then press (←)(PURGE).

You can't delete a backup object that you recalled to the stack—you get the `Object in Use` message. If you delete the object from the stack or store the object in a variable, then you can delete the backup object.

To search all ports for an object: 34

1. Enter the backup identifier for the object—*except* use & for the port number. (Press (α) (←)(ENTER) to type &.)
2. Execute RCL, EVAL, or PURGE.

If you use the & "wildcard" character for the port number, the HP 48 searches ports 2, 1, 0, and then main memory for the backup object—it uses the first occurrence of the name.

Example: If you enter `:&:BPG1` and press (←)(PURGE), you delete the first occurrence of *BPG1* in port 2, 1, 0, or main memory.

To get a list of backup objects in a port:

- Enter the port number (0, 1, or 2) and press (←)(MEMORY) (NXT) PVARS.

The PVARS command returns two results. Level 1 indicates the type of memory contained in the port: `"ROM"` (application card), `"SYSRAM"` (merged memory), or a number (the number of available bytes in user memory for port 0, or in the port's independent memory for port 1 or 2). Level 2 contains a list of backup identifiers and library identifiers.

To remove an independent RAM card with its backup objects:

- Turn off the HP 48 and unplug the card—see "To remove a plug-in card" on page 34-7.

To copy backup objects from a card into another HP 48:

1. Turn off the HP 48 and install the card—see "Installing and Removing RAM and ROM Cards" on page 34-5.
2. Turn on the HP 48.
3. Recall the object to the stack—see "To recall a backup object to the stack" on page 34-16.

You can also transfer objects between two HP 48s using their infrared ports—see "Transferring Data between Two HP 48s" on page 33-8.

Backing Up All of Memory

You can back up and restore the contents of the entire *HOME* directory in a backup object. The *HOME* directory includes all variables, user key assignments, and alarms. You can also include all flag settings if you want.

You can also back up memory in a computer file. See "Backing Up All of HP 48 Memory" on page 33-14.

Caution While backing up memory, make sure the ticking clock is not in the display. If the clock is in the display, it may corrupt the backup data.

To back up all of user memory in a backup object:

1. Optional: To back up flags settings too, press [→] [MODES] [NXT] RCLF, enter a variable name (with ' delimiters), and press [STO].
2. Enter a backup specifier for the backup object to create.
3. Press [←] [MEMORY] [NXT] [NXT] ARCHI.

ARCHIVE backs up only user memory—it does *not* back up independent memory.

Caution

Executing RESTORE *overwrites* the entire contents of user memory with the contents of the backup object. To save the stack, you can save it in another backup object—see "Saving and Restoring the Stack" on page 5-3.

To restore HP 48 user memory from a backup object:

1. Recall the backup object—see "To recall a backup object to the stack" on page 34-16.
2. Press ⬅ MEMORY NXT NXT RESTO.
3. Optional: To restore flag settings previously saved, recall the contents of variable containing the flag data and press ➡ MODES NXT STOF.

Example: To create in port 2 the backup object *JUN12* for all of user memory, enter `:2:JUN12` and execute ARCHIVE. To then restore user memory from the backup object, enter `:2:JUN12` and execute RESTORE.

Using Library Objects

A *library* is an object that contains named objects that can act as extensions to the built-in command set. The primary use of a library is to serve as a vehicle for a ROM- or RAM-based application. A ROM-based library resides in a plug-in application card (such as the HP Solve Equation Library Application Card) and is installed by inserting the card into port 1 or 2. (The HP 48S has no plug-in ports.) A RAM-based library can reside in a plug-in RAM card, or it can be transferred into user memory from the infrared or serial I/O port. (See the library's documentation for details).

Libraries offer several advantages over programs:

- Applications you write are protected from copying because the contents of a library can't be viewed, edited, or recalled to the stack.
- Libraries offer faster access to the variables used by applications.
- You can designate variables used in applications as "hidden" (unnamed) variables, which avoids cluttering the library's menu.

Creating Libraries

You can't create a library directly on the HP 48. But you can create one on a computer and load it into the HP 48.

To create a library using a computer:

1. **HP 48.** Create a directory of the objects you want contained in the library.
2. **HP 48 and Computer.** Transfer the directory variable to the computer—see "Transferring Data between a Computer and HP 48" on page 33-10.
3. **Computer.** Execute a library-building program named USRLIB that resides on your computer—see below.
4. **HP 48 and Computer.** Transfer the library object to the HP 48.

USRLIB is available electronically on the HP Calculator Bulletin Board system—see the inside back cover.

Setting Up Libraries

To set up a library:

34

1. Install the library in a port:
   - For an application card library, *turn off the HP 48* and insert the card into port 1 or 2.
   - For a RAM-based library, store it in independent memory (port 0, 1, or 2).
2. Attach the library:
   - For certain libraries, just turn the HP 48 off and on.
   - For other libraries, attach it manually—see below.

To use a library, it must be installed in a port and attached to a directory in user memory. The attachment may happen automatically when you install an application card—or you may have to do it yourself.

To store a RAM-based library in independent memory:

1. Put the library object on the stack. (Notice its library number and name.)
2. Enter the port number for storing the library (0, 1, or 2).
3. Press (STO).
4. Optional: Purge the original library object in user memory.

If you use port 0, the library is always available, even if you remove plug-in cards. If you use port 1 or 2, in must contain a RAM card set up as independent memory.

To manually attach a library that's in a port:

1. Change to the desired directory:
   - For access from all directories, change to the *HOME* directory.
   - For limited access, change to the desired directory. The library will be available only in this directory and its subdirectories.
2. Enter the *library identifier* for the library—it has the form :*port*:*number*. See below.
3. Press (⬅)(MEMORY) (NXT) ATTAC.

You can attach only one library to each directory—*except* you can attach any number to the *HOME* directory. (See the documentation that comes with the application card or RAM-based library for any other information about attaching the library.) 34

Each library is identified two ways:

- A *library identifier*, which has the form :*port*:*number*, where *number* is a unique number associated with the library. If you press (⬅)(LIBRARY) and PORT0, PORT1, or PORT2 for the port where you stored the library, the library number appears in the menu.
- A *library name*, which is a sequence of characters. If you press (⬅)(LIBRARY) in the directory where you attached the library or any subdirectory, the library name appears in the menu.

To delete a library:

1. Change to the directory where the library is attached.
2. Enter the *library identifier* for the library in independent memory—it has the form :*port*:*number*.
3. Press (ENTER) again to make a second copy of the library identifier.
4. Press (⬅)(MEMORY) (NXT) DETAC to detach it from the directory.
5. Press (⬅)(PURGE) to delete the library from independent memory.

Using Libraries

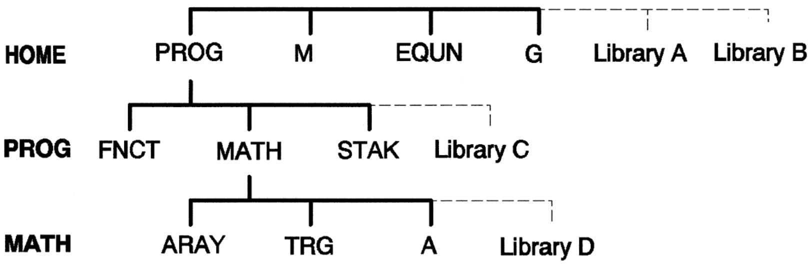
To get the menu of operations in a library:

1. If the library isn't attached to the *HOME* directory, change to the directory it's attached to—or to one of its subdirectories.
2. Press (⬅)(LIBRARY) and the menu key for the library name.

The LIBRARY menu contains the names of available libraries—libraries on the current directory path, not just in the current directory. The menu for an individual library contains the operations in that library. Press those menu keys to perform library operations.

Example: If you have the HP Solve Equation Library card installed, it's automatically attached to the *HOME* directory. Press
34 (⬅)(LIBRARY) EQLIB to display the menu of all the operations in the EQLIB library.

Example: Suppose your HP 48 has the following directory structure and attached libraries.



If you press (⬅)(LIBRARY) in the *HOME* directory, the menu includes A and B. If you press (⬅)(LIBRARY) in the *PROG* directory, the menu shows A, B, and C.

Summary of Library Commands

Library Commands

Key	Programmable Command	Description
STO	STO	Stores a library object from level 2 into independent memory in the port specified in level 1.
↱ RCL	RCL	Recalls the library object specified by the library identifier (:*port*:*number*) in level 1.
↰ PURGE	PURGE	Purges the RAM-based library specified by the library identifier (:*port*:*number*) in level 1.
↰ MEMORY (page 2):		
PVARS	PVARS	For the port number specified in level 1, returns to level 2 the list of the backup identifiers and library identifiers and to level 1 the type of memory: "ROM" (application card), "SYSRAM" (merged memory), or a number (the number of available bytes in user memory for port 0, or in the port's independent memory for port 1 or 2).
LIBS	LIBS	Displays a list containing the names, library numbers, and port numbers of all the libraries attached to the current directory.
ATTAC	ATTACH	Attaches to the current directory the library specified by the library number in level 1.
DETAC	DETACH	Detaches from the current directory the library specified by the library number in level 1.

34

Part 6

Appendixes

A

Support, Batteries, and Service

If Things Go Wrong

Whenever you run into problems—either following examples in this manual or solving your own problems—you can use these hints to get back on track. There's more information in the next section, "Answers to Common Questions," and in appendix B, "Messages."

If you want to clear a message:

- Press [ATTN] (the [ON] key).

If you get stuck in an unfamiliar condition:

- Press [ATTN] one or more times, until you see the normal stack display.

If you want to undo a mistake:

- To retrieve the last command line you executed (so you can change it and execute it again), press [←][LAST CMD] (above the [3] key).
- To remove the last result and get back the original data, press [←][LAST STACK] (above the [2] key).
- To keep the last result *and* get back the original data, press [→][LAST ARG] (above the [2] key).

If your command line has invalid syntax:

- Try to figure out what's wrong with the text in the command line—especially at the ▮ marker—then edit the line and press [ENTER].

  or
- Press [ATTN] [ATTN] to start over.

A

If you need to reset all calculator operating modes:

1. Press [ATTN] [↱] [#] 1008 [α] [↰] D (to get #1008d).
2. Press [↱] [CST] [NXT] STOF .

If you need to reset the calculator (and erase all memory):

1. *If there's anything in memory you want to keep, don't reset the calculator.*
2. Press and hold [ON].
3. At the same time, press the left and right menu keys (A and F), then release them.
4. Release the [ON] key.
5. Press NO .

The above steps also erase the contents of a plug-in RAM card—but *only* if its RAM is merged with the calculator's main memory.

If the calculator won't turn on:

1. Hold down [ON] and press [+] several times to check for too light of a display.
2. Install three new AAA batteries, as described under "Changing Batteries" on page A-7.
3. Check the calculator. See "Testing Calculator Operation" on page A-11.

If you think your calculator needs to be repaired:

1. Check its operation. See "Testing Calculator Operation" on page A-11.
2. Contact the HP Calculator Support department. See the inside back cover.
3. Send it to Hewlett-Packard. See "If the Calculator Requires Service" on page A-18.

A

Answers to Common Questions

You can obtain answers to questions about using your calculator from our Calculator Support department. Our experience has shown that many customers have similar questions about our products, so we've provided this section. If you don't find the answer to your question here, contact us at the address or phone number on the inside back cover.

Q: *I'm not sure whether the calculator is malfunctioning or if I'm doing something incorrectly. How can I verify that the calculator is operating properly?*

A: See "Testing Calculator Operation" on page A-11.

Q: *The ((•)) annunciator stays on even when the calculator is turned off. Is anything wrong?*

A: This indicates a low-battery condition in the calculator or a RAM card, or an alarm that is past due. To determine what is causing the ((•)) annunciator to stay on, turn the calculator off and then on. A message in the display will identify the problem. See "When to Replace Batteries" on page A-6 or "Setting Alarms" on page 24-5.

Q: *How can I determine how much memory is left in the calculator?*

A: Press ⬑ MEMORY MEM. The number of bytes of available memory will appear at the lower right corner of the display.
For exmaple, an empty memory for the HP 48SX should show approximately 30000 bytes of internal RAM (with no RAM cards installed).

Q: *How do I change the number of decimal places the HP 48 displays?*

A: Perform the following steps (see "Setting the Display Mode" on page 2-14):

1. Go to page 1 of the MODES menu: press ⬑ MODES.
2. Press the number of decimal places you want (0 to 11).
3. Press the menu key for the display format you desire: FIX, SCI, or ENG.

A

Q: *My numbers contain commas as decimal points. How do I restore periods?*

A: Perform the following steps:

1. Go to page 4 of the MODES menu: press ⬅ (MODES) (NXT) (NXT) (NXT).
2. Press the FM, menu key. (The label shows ■ only when comma is the separator.)

Q: *What does an* E *in a number mean (for example,* 2.51E-13*)?*

A: Exponent of 10 (for example, 2.51×10^{-13}). See "Keying In Numbers" on page 2-6 and "Setting the Display Mode" on page 2-14.

Q: *Why do trig functions give me unexpected results?*

A: The angle mode may be wrong for your problem. Check the angle mode annunciator: RAD means radians, GRAD means grads, and none means degrees. Press ⬅ (RAD) or use the ⬅ (MODES) menu to change the angle mode.

Q: *When I take the sine of π in Degrees mode, why do I get* 'SIN(π)' *instead of a number?*

A: The calculator is in Symbolic Result mode; 'SIN(π)' is the symbolic answer. Press ➡ (→NUM) to convert 'SIN(π)' to its numeric equivalent of .0548 … up to 11 decimal places (sin 3.14°). You can also press SYM■ on page 1 of the MODES menu to change to Numeric Results mode and prevent symbolic evaluation.

Q: *What does "object" mean?*

A: "Object" is the general term for all elements of data the HP 48 works with. Numbers, expressions, arrays, programs, and so on, are all types of objects. See chapter 4, "Objects," for a description of the object types accepted by the calculator.

Q: *What do three dots (...) mean at either end of a display line?*

A: The three dots (called an *ellipsis*) indicate that the displayed object is too long to display on one line. To view undisplayed portions of the object, use the ◀ or ▶ cursor keys.

Q: *How do I turn off the* HALT *annunciator?*

A: Press (PRG) CTRL KILL.

Q: *The calculator beeps and displays* `Bad Argument Type`. *What's wrong?*

A: The objects on the stack aren't the correct type for the command you are attempting. For example, executing `→UNIT` (in page 2 of the PRG OBJ menu) with a number in stack levels 1 and 2 causes this error.

Q: *The calculator beeps and displays* `Too Few Arguments`. *What's wrong?*

A: There are fewer arguments on the stack than required by the command you are attempting. For example, executing [+] with only one argument or number on the stack causes this error.

Q: *The calculator beeps and displays a message different from the two listed above. How do I find out what's wrong?*

A: Refer to appendix B, "Messages."

Q: *I can't find some variables that I used earlier. Where did they go?*

A: You may have been using the variables in a different directory. If you can't remember which directory you were using, you'll need to check all the directories in your calculator.

Q: *Sometimes my HP 48 seems to pause momentarily during a calculation. Is anything wrong?*

A: Nothing is wrong. The calculator does some system cleanup from time to time to eliminate temporary objects created from normal operation. This cleanup process frees memory for current operations. This happens less often if you make more memory available.

Q: *During normal operation, the printer prints several lines quickly, then slows down. Why?*

A: The calculator quickly transmits a certain amount of data to the printer, then slows its transmission rate to ensure that the printer can keep up.

Q: *How can I increase the printing speed of my HP 82240B Infrared Thermal Printer?*

A: Use an ac adapter with your HP 82240B printer so that the printer can print faster. Also, set the calculator delay to match the print speed (see "To change the delay between printed lines" on page 32-7).

Environmental Limits

To maintain product reliability, avoid getting the calculator and plug-in cards wet and observe the following temperature and humidity limits:

Calculator:

- Operating temperature: 0° to 45°C (32° to 113°F).
- Storage temperature: −20 to 65 °C (−4 to 149 °F).
- Operating and storage humidity: 90% relative humidity at 40 °C (104 °F) maximum.

Plug-In Cards:

- Operating temperature: 0 to 45 °C (32 to 113 °F).
- Storage temperature: −20 to 60 °C (−4 to 140 °F).
- Storage temperature for RAM card data retention: 0 to 60 °C (32 to 140 °F).
- Operating and storage humidity: 90% relative humidity at 40 °C (104 °F) maximum.

When to Replace Batteries

When a low-battery condition exists, the ((•)) annunciator remains on, even when the calculator is turned off. When the calculator is turned on during a low-battery condition, `Warning: LowBat( )` is displayed for approximately 3 seconds:

A

`LowBat(P1)` refers to port 1.
`LowBat(P2)` refers to port 2.
`LowBat(S)` refers to the calculator (system) batteries.

Replace the RAM card battery or the calculator batteries as soon as possible after the ((•)) low-battery annunciator and warning message appear. If you continue to use the calculator while the ((•)) annunciator is on, the display will eventually dim and you may lose calculator and RAM card data.

Under typical use, a RAM card's battery should last between 1 and 3 years. Be sure to mark the card with the battery-installation date, and, in case the RAM card is not in the calculator when the battery needs replacement, set an alarm for 1 year from that date to remind you to install a fresh battery. RAM cards do not come with a battery installed.

Changing Batteries

The HP 48 uses the following kinds of batteries:

- **Calculator Batteries.** Any brand of size AAA batteries. *Be sure that all three batteries are of the same brand and type.* (The use of rechargeable batteries is not recommended because of their lower capacity and short low-battery warning time.)
- **Plug-In RAM Card Batteries.** 3-volt 2016 coin cell.

To replace calculator batteries, use the steps below. To replace RAM card batteries, see "To change a RAM card battery" on page A-9.

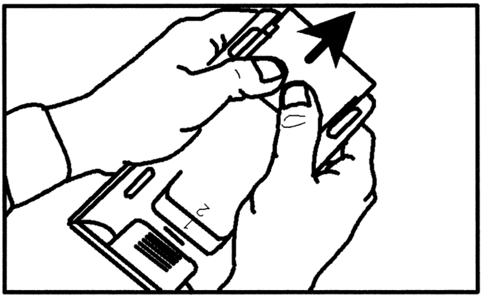
Caution

Whenever you remove batteries from the calculator, be sure the calculator is off and *do not press the* [ON] *key until the new batteries are installed.* If you press [ON] when batteries are not in the calculator, you may lose all of calculator memory.

To change calculator batteries:

1. Turn the calculator off. You may lose memory in the calculator and plug-in RAM cards if the calculator batteries are removed when the calculator is on.
2. Have three, fresh size AAA batteries (of the same brand and type) at hand. Wipe off both ends of each battery with a clean, dry cloth.

3. Remove the calculator battery-compartment door by pressing down and sliding it off away from the calculator. Be careful not to press the calculator's [ON] key. See the following illustration.

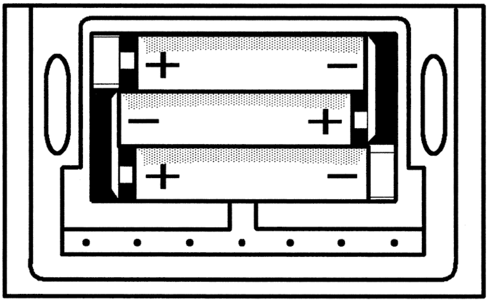


4. Turn the calculator over and shake the batteries out. After the batteries are out, you should replace them with fresh batteries within 2 minutes to protect against memory loss.

Warning **Do not mutilate, puncture, or dispose of batteries in fire. The batteries can burst or explode, releasing hazardous chemicals. Discard used batteries according to the manufacturer's instructions.**

A

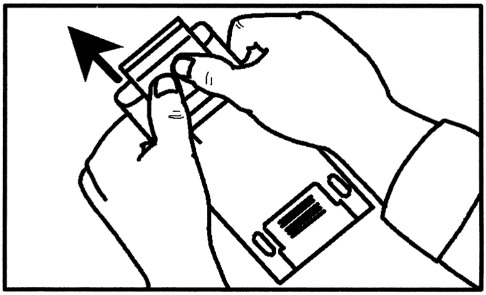
5. Position the batteries according to the outlines in the bottom of the battery compartment. *Avoid touching the battery terminals.* Batteries are easier to install if the negative (plain) ends are inserted first, and if the center battery is installed last. See the following illustration.



6. Replace the battery-compartment door by sliding the tabs on the door into the slots in the calculator case.
7. Press [ON] to turn the calculator on.

To change a RAM card battery:

1. Turn the calculator over and remove the plastic cover over the plug-in card ports (on the display-end of the calculator).



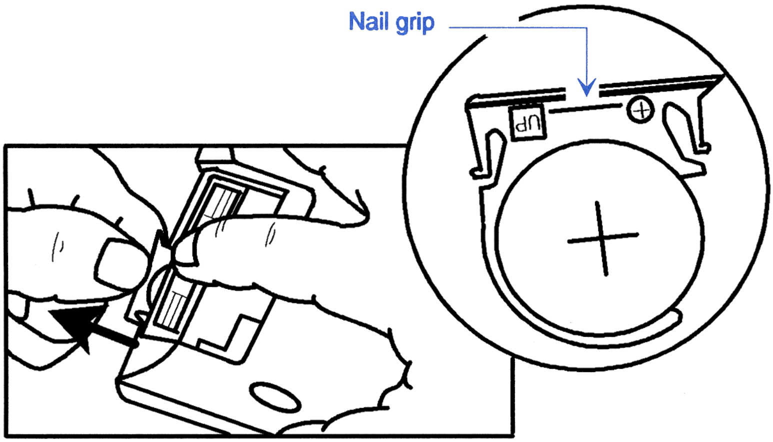
2. With the RAM card in port 1 or 2, *turn on the calculator*.

Caution

Make sure you *turn on the calculator* before you change a RAM card battery. RAM cards run off the calculator batteries only while the calculator is on. RAM memory may be lost if you remove a RAM card battery while the calculator is off or while the card is not installed in the calculator.

---

3. Place your index finger in the recess near the exposed end of the RAM card—this prevents removal of the card from the calculator when you remove the card's battery holder. Now insert the thumbnail of your free hand into the nail grip in the black plastic at the left side of the end of the card and pull the battery holder out of the card.



A

4. Remove the old battery from the plastic battery holder.

Warning

Do not mutilate, puncture, or dispose of batteries in fire. The batteries can burst or explode, releasing hazardous chemicals. Discard used batteries according to the manufacturer's instructions.

---

5. Install a fresh, 3-volt 2016 coin cell in the plastic battery holder and reinsert the holder (with battery) into the card. *Be sure to*

install the battery with the side marked "+" toward the front of the card.

6. Mark the card with the battery-installation date, and set an alarm for 1 year from that date to remind you to change it. (If you unplug the card, the HP 48 can't check the card's battery level.)
7. Replace the plug-in port cover.

Testing Calculator Operation

Use the following guidelines to determine whether the calculator is functioning properly. Test the calculator after every step to see if operation has been restored. If your calculator requires service, see "If the Calculator Requires Service" on page A-18.

If the calculator won't turn on or doesn't respond when you press the keys:

1. Make sure that three fresh batteries are correctly installed in the calculator.
2. Press and release [ON].
3. If the display is blank, press and hold [ON]; press and release [+] several times until characters become visible; then release [ON]. If no characters appear in the display, the calculator requires service.
4. If a halted program won't respond when you press [ATTN], try pressing [ATTN] again.
5. If the keyboard is "locked," perform a system halt:
   a. Press and hold [ON].
   b. Press and release the "C" key (the key with C next to it).
   c. Release [ON]. The empty stack display should appear.
6. If the problem still exists, perform a memory reset:
   a. Press and hold [ON].
   b. Press and hold the "A" and "F" keys (the keys with A and F next to them).
   c. Release all three keys.

A

The calculator will beep and display the message Try To Recover Memory? at the top of the display. Press YES to recover as much memory as possible.

If these steps fail to restore operation, the calculator requires service.

If the calculator responds to keystrokes, but you suspect it's malfunctioning:

1. Run the self-test described in the next section.
   - If the calculator fails the self-test, it requires service.
   - If the calculator passes the self-test, you may have made a mistake operating the calculator. Reread appropriate portions of the manual and check "Answers to Common Questions" on page A-3.
2. Contact the Calculator Support department. The address and phone number are listed on the inside back cover.

Self-Test

If the display turns on, but the calculator does not seem to be operating properly, run the diagnostic self-test.

To run the self-test:

1. Turn on the calculator.
2. Press and hold [ON].
3. Press and release the "E" key (the key with E next to it).
4. Release [ON].

A

The diagnostic self-test tests the internal ROM and RAM, and generates various patterns in the display. The test repeats continuously until you perform a system halt.

To halt the self-test (system halt):

1. Press and hold [ON].
2. Press and release the "C" key (the key with C next to it).
3. Release [ON]. The empty stack display should appear.

If the self-test indicates an internal ROM or RAM failure (if IROM OK and IRAM OK are not displayed), the calculator requires service.

The diagnostic self-test should be successfully completed before running any of the tests described in the following sections.

Keyboard Test

This test checks all of the calculator's keys for proper operation.

To run the interactive keyboard test:

1. Turn on the calculator.
2. Press and hold [ON].
3. Press and release the "D" key (the key with D next to it).
4. Release [ON].
5. Press and release the "E" key (the key with E next to it). KBD1 will appear in the upper left corner of the display.
6. Starting at the upper left corner and moving left to right, press each of the 49 keys on the keyboard.

If you press the keys in the proper order and they're functioning properly, the calculator emits a high-pitch beep at each press of a key. When you press the 49th key, [+], the displayed message should change to KBD1 OK.

If you press a key out of sequence, a five-digit hexadecimal number will appear next to KBD1. Reset the keyboard test (do steps 1 through 3 above), and rerun the test.

If a key isn't functioning properly, the next keystroke displays the hex location of the expected and the received location. If you pressed the keys in order and got this message, the calculator requires service. Be sure to include a copy of the error message when you ship the calculator for service.

To exit the keyboard test (system halt):

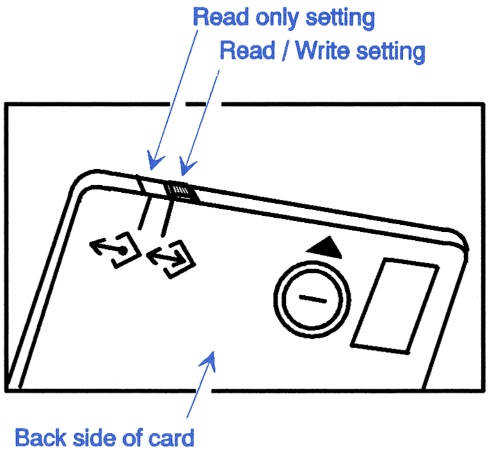
1. Press and hold [ON].
2. Press and release the "C" key (the key with C next to it).
3. Release [ON]. The empty stack display should appear.

Port RAM Test

The port RAM test nondestructively tests the ports and the installed plug-in RAM cards. (Plug-in RAM-card memory is preserved.)

To run the port RAM test:

1. Check that a plug-in RAM card is properly installed in port 1 or port 2.
2. Verify that the switch on each card is set to the "Read/Write" position.



A

3. Turn on the calculator.
4. Press and hold [ON].
5. Press and release the "D" key (the key with D next to it).
6. Release [ON]. A vertical line will appear at both sides and at the center of the display.
7. Press and release [▲].

RAM1 or RAM2 will appear at the top left corner of the display and the size of the corresponding plug-in RAM card (32K or 128K) will appear at the top right corner of the display. OK will appear to the right of RAM1 or RAM2 when the port RAM test has been successfully completed.

A failure message (for example, RAM1 00002) will be displayed for each port that does not contain a plug-in RAM card or if a card's read/write switch is in the "write-protect" position. This message should be ignored.

If OK doesn't appear for a RAM card set to read/write, the card should be moved to the other port and the test rerun. If OK still doesn't appear, the RAM card should be replaced with a new one.

To return to normal calculator operation (system halt):

1. Press and hold [ON].
2. Press and release the "C" key (the key with C next to it).
3. Release [ON]. The empty stack display should appear.

IR Loop-Back Test

This test checks the operation of the send and receive infrared sensors and their associated circuits.

To run the IR loop-back test:

1. Turn on the calculator.
2. Press and hold [ON].
3. Press and release the "D" key (the key with D next to it).
4. Release [ON]. A vertical line will appear at both sides and at the center of the display.
5. Be sure that the plastic plug-in card cover is in place and that it covers the clear lamp bulbs in the top end of the calculator.
6. Press [EVAL].

IRLB will appear at the top left corner of the display. If OK appears to the right of IRLB, the calculator passes this test. If OK doesn't appear, the calculator requires service.

To return to normal calculator operation (system halt):

1. Press and hold [ON].
2. Press and release the "C" key (the key with C next to it).
3. Release [ON]. The empty stack display should appear.

Serial Loop-Back Test

This test checks the operation of the send and receive circuits of the serial interface at the top of the calculator.

To run the serial loop-back test:

1. Turn on the calculator.
2. Press and hold ON.
3. Press and release the "D" key (the key with D next to it).
4. Release ON. A vertical line will appear at both sides and at the center of the display.
5. Temporarily connect (short) the middle two pins (pins 2 and 3) of the 4-pin serial connector at the top end of the calculator. Be careful not to bend or severely jar the pins.
6. Press PRG.

U_LB will appear at the top left corner of the display. If OK appears to the right of U_LB, the calculator passes this test. If OK doesn't appear, the calculator requires service.

Note

If you inadvertently short pins 1 and 2 or pins 3 and 4 of the serial connector, the loop-back test will return U_LB 00001 or U_LB 00002 (test-failed message), but you will not damage the calculator.

To return to normal calculator operation (system halt):

1. Press and hold ON.
2. Press and release the "C" key (the key with C next to it).
3. Release ON. The empty stack display should appear.

Limited One-Year Warranty

What Is Covered. *The calculator* (except for the batteries, or damage caused by the batteries) *and calculator accessories are warranted by Hewlett-Packard against defects in materials and workmanship for one year from the date of original purchase.* If you sell your unit or give it as a gift, the warranty is automatically transferred to the new owner and remains in effect for the original one-year period. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to a Hewlett-Packard service center. (Replacement may be made with a newer model of equal or better functionality.)

This warranty gives you specific legal rights, and you may also have other rights that vary from state to state, province to province, or country to country.

What Is Not Covered. *Batteries, and damage caused by the batteries, are not covered by the Hewlett-Packard warranty. Check with the battery manufacturer about battery and battery leakage warranties.*

Damage caused to the HP 48 as the result of using nonapproved plug-in cards and plug-in accessories is not covered by the Hewlett-Packard warranty.

This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than an authorized Hewlett-Packard service center.

No other express warranty is given. The repair or replacement of a product is your exclusive remedy. **ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY.** Some states, provinces, or countries do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. **IN NO EVENT SHALL HEWLETT-PACKARD COMPANY BE LIABLE FOR CONSEQUENTIAL DAMAGES.** Some states, provinces, or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Products are sold on the basis of specifications applicable at the time of manufacture. Hewlett-Packard shall have no obligation to modify or update products, once sold.

Consumer Transactions in the United Kingdom. This warranty shall not apply to consumer transactions and shall not affect the statutory rights of a consumer. In relation to such transactions, the rights and obligations of Seller and Buyer shall be determined by statute.

If the Calculator Requires Service

Hewlett-Packard maintains service centers in many countries. These centers will repair a calculator, or replace it with the same model or one of equal or better functionality, whether it is under warranty or not. There is a service charge for service after the warranty period. Calculators normally are serviced and reshipped within 5 working days.

Note — If the contents of your calculator's memory are important, you should back up the memory on a plug-in RAM card, another HP 48, or a computer before sending in the calculator for repair.

- **In the United States:** Send the calculator to the Corvallis Service Center listed on the inside of the back cover.
- **In Europe:** Contact your Hewlett-Packard sales office or dealer, or Hewlett-Packard's European headquarters (address below) for the location of the nearest service center. *Do not ship the calculator for service without first contacting a Hewlett-Packard office.*

  Hewlett-Packard S.A.
  150, Route du Nant-d'Avril
  P.O. Box CH 1217 Meyrin 2
  Geneva, Switzerland
  Telephone: 022 780.81.11

- **In other countries:** Contact your Hewlett-Packard sales office or dealer or write to the Corvallis Service Center (listed on the inside of the back cover) for the location of other service centers. If local service is unavailable, you can ship the calculator to the Corvallis Service Center for repair.

All shipping, reimportation arrangements, and customs costs are your responsibility.

Service Charge. Contact the Corvallis Service Center (inside back cover) for the standard out-of-warranty repair charges. This charge is subject to the customer's local sales or value-added tax wherever applicable.

Calculator products damaged by accident or misuse are not covered by the fixed charges. These charges are individually determined based on time and material.

Shipping Instructions. If your calculator requires service, ship it to the nearest authorized service center or collection point.

- Include your return address and a description of the problem.
- Include proof of purchase date if the warranty has not expired.
- Include a purchase order, check, or credit card number plus expiration date (VISA or MasterCard) to cover the standard repair charge.
- Ship your calculator postage *prepaid* in adequate protective packaging to prevent damage. Shipping damage is not covered by the warranty, so we recommend that you insure the shipment.

Warranty on Service. Service is warranted against defects in materials and workmanship for 90 days from the date of service.

Service Agreements. In the U.S., a support agreement is available for repair and service. For additional information, contact the Corvallis Service Center (see the inside of the back cover).

B

Messages

This appendix lists selected HP 48 messages. In the tables below, messages are first arranged alphabetically by content and then numerically by message number.

Messages Listed Alphabetically

Message	Meaning	# (hex)
`Acknowledged`	Alarm acknowledged.	619
`Alarm`	Alarm not acknowledged yet.	(none)
`Autoscaling`	Calculator is autoscaling x- and/or y- axis.	610
`Awaiting Server Cmd.`	Indicates Server mode active.	C0C
`Bad Argument Type`	One or more stack arguments were incorrect type for operation.	202
`Bad Argument Value`	Argument value out of operation's range.	203
`Bad Guess(es)`	Guess(es) supplied to HP Solve application or ROOT lie outside domain of equation.	A01
`Bad Packet Block check`	Computed packet checksum doesn't match checksum in packet.	C01
`Can't Edit Null Char.`	Attempted to edit a string containing character with code 0.	102

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Circular Reference	Attempted to store a variable name into itself.	129
Connecting	Indicates verifying IR or serial connection.	C0A
Constant?	HP Solve application or ROOT returned same value at every sample point of current equation.	A02
Copied to stack	→STK copied selected equation to stack.	623
Current equation:	Identifies current equation.	608
Deleting Column	MatrixWriter application is deleting a column.	504
Deleting Row	MatrixWriter application is deleting a row.	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
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

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
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 MatrixWriter application, DRAW, or HP Solve application active.	126
I/O setup menu	Identifies I/O setup menu.	61C
Implicit () off	Implicit parentheses off.	207
Implicit () on	Implicit parentheses on.	208
Incomplete Subexpression	▶, ▼, or ENTER pressed before all function arguments supplied.	206
Inconsistent Units	Attempted unit conversion with incompatible units.	B02
Infinite Result	Math exception: Calculation such as 1/0 has infinite result.	305
Inserting Column	MatrixWriter application is inserting a column.	504
Inserting Row	MatrixWriter application is inserting a row.	503
Insufficient Memory	Not enough free memory to execute operation.	001
Insufficient Σ Data	A Statistics command was executed when *ΣDAT* did not contain enough data points for calculation.	603
Interrupted	The HP Solve application or ROOT was interrupted by ATTN.	A03

B

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Invalid Array Element	ENTER returned object of wrong type for current matrix.	502
Invalid Card Data	HP 48 does not recognize data on plug-in card.	008
Invalid Date	Date argument not real number in correct format, or was out of range.	D01
Invalid Definition	Incorrect structure of equation argument for DEFINE.	12C
Invalid Dimension	Array argument had wrong dimensions.	501
Invalid EQ	Attempted operation from GRAPHICS FCN menu when *EQ* did not contain algebraic, or, attempted DRAW with CONIC plot type when *EQ* did not contain algebraic.	607
Invalid IOPAR	*IOPAR* not a list, or one or more objects in list missing or invalid.	C12
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

B

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Invalid Repeat	Alarm repeat interval out of range.	D03
Invalid Server Cmd.	Invalid command received while in Server mode.	C08
Invalid Syntax	HP 48 unable to execute [ENTER] 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)	Nonlinear curve fit attempted when *ΣDAT* matrix contained a negative element.	605
Invalid Σ Data LN(0)	Nonlinear 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
LAST CMD Disabled	[LAST CMD] pressed while that recovery feature disabled.	125
LAST STACK Disabled	[LAST STACK] pressed while that recovery feature disabled.	124

B

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
LASTARG Disabled	LASTARG executed while that recovery feature disabled.	205
LowBat( )	Replace calculator batteries (S), or replace plug-in card batteries (P1) or (P2).	(none)
Low Battery	System batteries too low to safely print or perform I/O.	C14
Memory Clear	HP 48 memory was cleared.	005
Name Conflict	The | (where) function attempted to assign a value to the 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 result, between 0 and −MINR.	302
No Current Equation	SOLVR, 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 specified RAM port.	00B
No Room to Save Stack	Not enough free memory to save copy of the stack. LAST STACK is automatically disabled.	101

B

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
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
Non-Empty Directory	Attempted to purge nonempty directory.	12B
Non-Real Result	Execution of HP Solve application, ROOT, DRAW, or $\int$ returned result other than real number or unit.	12F
Nonexistent Alarm	Alarm list did not contain alarm specified by alarm command.	D04
Nonexistent ΣDAT	Statistics command executed when ΣDAT did not exist.	602
Object Discarded	Sender sent an EOF (Z) packet with a "D" in the data field.	C0F
Object In Use	Attempted PURGE or STO into a backup object when its stored object was in use.	009
Object Not in Port	Attempted to access a nonexistent backup object or library.	00C
<OFF SCREEN>	Function value, root, extremum, or intersection was not visible in current display.	61F
Out of Memory	One or more objects must be purged to continue calculator operation.	135

B

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Overflow	Math exception: Calculation returned result greater in absolute value than MAXR.	303
Packet #	Indicates packet number during send or receive.	C10
Parity Error	Received bytes' parity bit doesn't match current parity setting.	C05
Port Closed	Possible IR or serial hardware failure. Run self-test.	C09
Port Not Available	Used a port command on an empty or nonexistent port, or one containing ROM instead of RAM. (Ports 1 and 2 don't exist in the HP 48S.) Attempted to execute a server command that itself uses the I/O port.	00A
Positive Underflow	Math exception: Calculation returned positive result, between 0 and MINR.	301
Power Lost	Calculator turned on following a power loss. Memory may have been corrupted.	006
Processing Command	Indicates processing of host command packet.	C11
Protocol Error	Received a packet whose length was shorter than a null packet. Maximum packet length parameter from other machine is illegal.	C07

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Receive Buffer Overrun	Kermit: More than 255 bytes of retries sent before HP 48 received another packet. SRECV: Incoming data overflowed the buffer.	C04
Receive Error	UART overrun or 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
Timeout	Printing to serial port: Received XOFF and timed out waiting for XON. Kermit: Timed out waiting for packet to arrive.	C02

B

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Too Few Arguments	Command required more arguments than were available on stack.	201
Transfer Failed	10 successive attempts to receive a good packet were unsuccessful.	C06
Unable to Isolate	ISOL failed because specified name absent or contained in argument of function with no inverse.	130
Undefined Local Name	Executed or recalled local name for which corresponding local variable did not exist.	003
Undefined Name	Executed or recalled global name for which corresponding variable does not exist.	204
Undefined Result	Calculation such as 0/0 generated mathematically undefined result.	304
Undefined XLIB Name	Executed an XLIB name when specified library absent.	004
Wrong Argument Count	User-defined function evaluated with an incorrect number of parenthetical arguments.	128
x and y-axis zoom.	Identifies zoom option.	627
x axis zoom.	Identifies zoom option.	625
x axis zoom w/AUTO.	Identifies zoom option.	624
y axis zoom.	Identifies zoom option.	626

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
ZERO	Result returned by the HP Solve application or ROOT is a root (a point at which current equation evaluates to zero).	A04
""	Identifies no execution action when EXECS pressed.	61E

B

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
008	Invalid Card Data
009	Object In use
00A	Port Not available
00B	No Room in Port
00C	Object Not in Port
101	No Room to Save Stack
102	Can't Edit Null Char.
103	Invalid User Function
104	No Current Equation
106	Invalid Syntax
124	LAST STACK Disabled
125	LAST CMD Disabled
126	HALT Not Allowed
128	Wrong Argument Count
129	Circular Reference
12A	Directory Not Allowed
12B	Non-Empty Directory
12C	Invalid Definition
12E	Invalid PPAR
12F	Non-Real Result
130	Unable to Isolate
131	No Room to Show Stack
Out-of-Memory Prompts	
135	Out of Memory
13C	Name Conflict

B

Messages Listed Numerically (continued)

# (hex)	Message
Stack Errors	
201	Too Few Arguments
202	Bad Argument Type
203	Bad Argument Value
204	Undefined Name
205	LASTARG Disabled
EquationWriter Application Messages	
206	Incomplete Subexpression
207	Implicit () off
208	Implicit () on
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)

B

Messages Listed Numerically (continued)

# (hex)	Message
Plot, I/O, Time, and HP Solve 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`
60F	`No Statistics data to plot`
610	`Autoscaling`
614	`Select a model`
619	`Acknowledged`
61A	`Enter alarm, press SET`
61B	`Select repeat interval`
61C	`I/O setup menu`
61D	`Plot type:`
61E	`""`
61F	`(OFF SCREEN)`
620	`Invalid PTYPE`
621	`Name the stat data, press ENTER`
622	`Enter value (zoom out if >1), press ENTER`
623	`Copied to stack`
624	`x axis zoom w/AUTO.`
625	`x axis zoom.`
626	`y axis zoom.`
627	`x and y-axis zoom.`
A01	`Bad Guess(es)`
A02	`Constant?`
A03	`Interrupted`
A04	`Zero`
A05	`Sign Reversal`
A06	`Extremum`

B

Messages Listed Numerically (continued)

# (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	I/O: Batt Too Low
C15	Empty Stack
C17	Invalid Name
Time Messages	
D01	Invalid Date
D02	Invalid Time
D03	Invalid Repeat
D04	Nonexistent Alarm
Miscellaneous Messages	
70000	(DOERR error)

B

C

HP 48 Character Codes

Except for character numbers 128 through 159, the HP 48 character set is based on the ISO 8859 Latin 1 character set.

You can type *most* of the HP 48 characters directly into the display from the Alpha keyboard—see the alpha-keyboard diagram on page 2-8. However, certain characters are *not* on the Alpha keyboard—to enter one of these characters, you refer to it by its character code. You can enter *any* HP 48 character this way.

To enter any character as a string:

1. Enter its character code.
2. Press (PRG) OBJ (NXT) (NXT) CHR (the CHR command).

To get the character code for the first character in a string:

1. Put the string in level 1.
2. Press (PRG) OBJ (NXT) (NXT) NUM (the NUM command).

If there's a character you use frequently that isn't available on the primary or alpha keyboards, you can assign the character to the user keyboard for easy access—see "Assigning User Keys" on page 15-6.

C

Character Codes

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
0	■	32		64	@	96	`
1	■	33	!	65	A	97	a
2	■	34	"	66	B	98	b
3	■	35	#	67	C	99	c
4	■	36	$	68	D	100	d
5	■	37	%	69	E	101	e
6	■	38	&	70	F	102	f
7	■	39	'	71	G	103	g
8	■	40	(	72	H	104	h
9	■	41	)	73	I	105	i
10	■	42	*	74	J	106	j
11	■	43	+	75	K	107	k
12	■	44	,	76	L	108	l
13	■	45	-	77	M	109	m
14	■	46	.	78	N	110	n
15	■	47	/	79	O	111	o
16	■	48	0	80	P	112	p
17	■	49	1	81	Q	113	q
18	■	50	2	82	R	114	r
19	■	51	3	83	S	115	s
20	■	52	4	84	T	116	t
21	■	53	5	85	U	117	u
22	■	54	6	86	V	118	v
23	■	55	7	87	W	119	w
24	■	56	8	88	X	120	x
25	■	57	9	89	Y	121	y
26	■	58	:	90	Z	122	z
27	■	59	;	91	[	123	{
28	■	60	<	92	\	124	|
29	■	61	=	93	]	125	}
30	■	62	>	94	^	126	~
31	■	63	?	95	_	127	▒

Character Codes (continued)

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
128	∡	160		192	À	224	à
129	x̄	161	¡	193	Á	225	á
130	∇	162	¢	194	Â	226	â
131	√	163	£	195	Ã	227	ã
132	∫	164	¤	196	Ä	228	ä
133	Σ	165	¥	197	Å	229	å
134	▶	166	¦	198	Æ	230	æ
135	π	167	§	199	Ç	231	ç
136	∂	168	¨	200	È	232	è
137	≤	169	©	201	É	233	é
138	≥	170	ª	202	Ê	234	ê
139	≠	171	«	203	Ë	235	ë
140	α	172	¬	204	Ì	236	ì
141	→	173	-	205	Í	237	í
142	←	174	®	206	Î	238	î
143	↓	175	¯	207	Ï	239	ï
144	↑	176	°	208	Ð	240	ð
145	γ	177	±	209	Ñ	241	ñ
146	δ	178	²	210	Ò	242	ò
147	ε	179	³	211	Ó	243	ó
148	η	180	´	212	Ô	244	ô
149	θ	181	µ	213	Õ	245	õ
150	λ	182	¶	214	Ö	246	ö
151	ρ	183	·	215	×	247	÷
152	σ	184	¸	216	Ø	248	ø
153	τ	185	¹	217	Ù	249	ù
154	ω	186	º	218	Ú	250	ú
155	Δ	187	»	219	Û	251	û
156	Π	188	¼	220	Ü	252	ü
157	Ω	189	½	221	Ý	253	ý
158	■	190	¾	222	Þ	254	þ
159	∞	191	¿	223	ß	255	ÿ

C

D

Menu Numbers and Menu Maps

The following table lists the HP 48 built-in menus and the corresponding menu numbers. The menu number for a library is the same as the library number.

Menu Numbers and Names

No.	Name	No.	Name
0	Last Menu	20	MODES
1	CST	21	MODES Customization
2	VAR	22	MEMORY
3	MTH	23	MEMORY Arithmetic
4	MTH PARTS	24	LIBRARY
5	MTH PROB	25	PORT 0
6	MTH HYP	26	PORT 1
7	MTH MATR	27	PORT 2
8	MTH VECTR	28	EDIT
9	MTH BASE	29	SOLVE
10	PRG	30	SOLVE SOLVR
11	PRG STK	31	PLOT
12	PRG OBJ	32	PLOT PTYPE
13	PRG DISP	33	PLOT PLOTR
14	PRG CTRL	34	ALGEBRA
15	PRG BRCH	35	TIME
16	PRG TEST	36	TIME ADJST
17	PRINT	37	TIME ALRM
18	I/O	38	TIME ALRM RPT
19	I/O SETUP	39	TIME SET

Menu Numbers and Names (continued)

No.	Name	No.	Name
40	STAT	50	UNITS ENRG
41	STAT MODL	51	UNITS POWR
42	UNITS Catalog	52	UNITS PRESS
43	UNITS LENG	53	UNITS TEMP
44	UNITS AREA	54	UNITS ELEC
45	UNITS VOL	55	UNITS ANGL
46	UNITS TIME	56	UNITS LIGHT
47	UNITS SPEED	57	UNITS RAD
48	UNITS MASS	58	UNITS VISC
49	UNITS FORCE	59	UNITS Command

MTH

PARTS	PROB	HYP	MATR	VECTR	BASE

PARTS

ABS	SIGN	CONJ	ARG	RE	IM
MIN	MAX	MOD	%	%CH	%T
MANT	XPON	IP	FP	FLOOR	CEIL
RND	TRNC	MAXR	MINR		

PROB

COMB	PERM	!	RAND	RDZ	
UTPC	UTPF	UTPN	UTPT		

HYP

SINH	ASINH	COSH	ACOSH	TANH	ATAN
EXPM	LNP1				

MATR

CON	IDN	TRN	RDM	DET	RSD
ABS	RNRM	CNRM			

VECTR

XYZ ■	R∡Z	R∡∡	CROSS	DOT	ABS
V →	→V2	→V3	D→R	R→D	

BASE

HEX	DEC ■	OCT	BIN	STWS	RCWS
RL	RR	RLB	RRB	R→B	B→R
SL	SR	SLB	SRB	ASR	
AND	OR	XOR	NOT		

PRG

STK	OBJ	DSPL	CTRL	BRCH	TEST

STK

OVER	ROT	ROLL	ROLLD	PICK	DEPTH
DUP	DUP2	DUPN	DROP2	DRPN	

OBJ

OBJ→	EQ→	→ARR	→LIST	→STR	→TAG
R→C	C→R	DTAG	→UNIT	TYPE	VTYPE
SIZE	POS	REPL	SUB	NUM	CHR
PUT	GET	PUTI	GETI		

DSPL

PICT	PVIEW	LINE	TLINE	BOX	ARC
PIXON	PIXOF	PIX?	PX→C	C→PX	SIZE
→GRO	BLAN	GOR	GXOR	REPL	SUB
→LCD	LCD→	CLLCD	DISP	FREEZ	TEXT

CTRL

DBUG	SST	SST↓	NEXT	HALT	KILL
INPUT	PROM	DISP	MENU	WAIT	KEY
DOERR	ERRN	ERRM	ERR0	BEEP	OFF

BRCH

IF	CASE	START	FOR	DO	WHILE
THEN	END	NEXT	STEP	UNTIL	REPEA
ELSE	IFERR	IFT	IFTE		

TEST

AND	OR	XOR	NOT	SAME	TYPE
==	≠	<	>	≤	≥
SF	CF	FS?	FC?	FS?C	FC?C

D

MTH and PRG Menus

PRINT, I/O, MODES, MEMORY, GRAPH, MATRIX, EDIT, SOLVE Menus

⬅ PRINT

PR1	PRST	PRSTC	PRLCD	PRVAR	CR
DELAY	OLDPR				

⬅ I/O

SEND	RECV	SERVE	KGET	FINIS	SETUP
RECN	PKT	KERR		OPENI	CLOSE
XMIT	SRECV	STIME	SBRK	BUFLE	

SETUP

IR/W	ASCII	BAUD	PARIT	CKSM	TRAN

⬅ MODES

STD■	FIX	SCI	ENG	SYM■	BEEP■
STK■	ARG■	CMD■	CNC■	ML■	CLK
DEG■	RAD	GRAD	XYZ■	R∡Z	R∡∡
HEX	DEC■	OCT	BIN	FM,	

➡ MODES

ASN	STOK	RCLK	DELK	MENU	CST
TMEN	RCLM	STOF	RCLF	SF	CF
FS?	FC?	FS?C	FC?C		

⬅ MEMORY

MEM	BYTES	VARS	ORDER	PATH	CRDIR
TVARS	PVARS	NEWO	LIBS	ATTAC	DETAC
MERG	FREE	ARCHI	RESTO	PGDIR	

➡ MEMORY

STO+	STO−	STO*	STO/	INCR	DECR
SINV	SNEG	SCON			

⬅ GRAPH

ZOOM	Z-BOX	CENT	COORD	LABEL	FCN
DOT+	DOT−	LINE	TLINE	BOX	CIRCL
MARK	REPL	SUB	DEL	+/−	KEYS

ZOOM

XAUTO	X	Y	XY		EXIT

FCN

ROOT	ISECT	SLOPE	AREA	EXTR	EXIT
F(X)	F'	NXEQ			

➡ MATRIX

EDIT	VEC■	←WID	WID→	GO→■	GO↓
+ROW	−ROW	+COL	−COL	→STK	↑STK

⬅ EDIT

←SKIP	SKIP→	←DEL	DEL→	INS■	↑STK

⬅ SOLVE

SOLVR	ROOT	NEW	EDEQ	STEQ	CAT

CAT

PLOTR	SOLVR	EQ+	EDIT	→STK	VIEW
ORDER	PURG	FAST			

D

← PLOT

PLOTR	PTYPE	NEW	EDEQ	STEQ	CAT

PLOTR

ERASE	DRAW	AUTO	XRNG	YRNG	INDEP
DEPN	PTYPE	RES	CENT	SCALE	RESET
AXES	DRAX	LABEL	*H	*W	PDIM

PTYPE

FUNC	CONIC	POLAR	PARA	TRUTH	BAR
HIST	SCATT				

CAT

PLOTR	SOLVR	EQ+	EDIT	→STK	VIEW
ORDER	PURG	FAST			

→ PLOT

ERASE	DRAW	AUTO	XRNG	YRNG	INDEP
DEPN	PTYPE	RES	CENT	SCALE	RESET
AXES	DRAX	LABEL	*H	*W	PDIM

← ALGEBRA

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

→ ALGEBRA

PLOTR	SOLVR	EQ+	EDIT	→STK	VIEW
ORDER	PURG	FAST			

← TIME

SET	ADJST	ALRM	ACK	ACKA	CAT
DATE+	DDAYS	DATE	TIME	TSTR	TICKS
→HMS	HMS→	HMS+	HMS–		

SET

→DAT	→TIM	A/PM	12/24	M/D	

ADJST

HR+	HR–	MIN+	MIN–	SEC+	SEC–
CLKA					

ALRM

>DATE	>TIME	A/PM	EXEC	RPT	SET
STOAL	RCLAL	DELAL	FINDA		

RPT

WEEK	DAY	HOUR	MIN	SEC	NONE

CAT

PURG		EXECS	EDIT	→STK	VIEW

← STAT

Σ+	CLΣ	NEW	EDITΣ	STOΣ	CAT
TOT	MEAN	SDEV	MAXΣ	MINΣ	BINS
XCOL	YCOL	BARPL	HISTP	SCATR	ΣLINE
LR	PREDX	PREDY	CORR	COV	MODL
ΣX	ΣY	ΣX^2	ΣY^2	ΣX*Y	NΣ

MODL

LIN	LOG	EXP	PWR	BEST	

CAT

1-VAR	PLOT	2-VAR	EDIT	→STK	VIEW
ORDER	PURG				

→ STAT

TOT	MEAN	SDEV	MAXΣ	MINΣ	BINS
XCOL	YCOL	BARPL	HISTP	SCATR	ΣLINE
LR	PREDX	PREDY	CORR	COV	MODL
ΣX	ΣY	ΣX^2	ΣY^2	ΣX*Y	NΣ
Σ+	CLΣ	NEW	EDITΣ	STOΣ	CAT

D

PLOT, ALGEBRA, TIME, STAT Menus

⬅ UNITS

LENG	AREA	VOL	TIME	SPEED	MASS
FORCE	ENRG	POWR	PRESS	TEMP	ELEC
ANGL	LIGHT	RAD	VISC		

LENG

M	CM	MM	YD	FT	IN
MPC	PC	LYR	AU	KM	MI
NMI	MIUS	CHAIN	RD	FATH	FTUS
MIL	μ	A	FERMI		

AREA

M^2	CM^2	B	YD^2	FT^2	IN^2
KM^2	HA	A	MI^2	MIUS^	ACRE

VOL

M^3	ST	CM^3	YD^3	FT^3	IN^3
L	GALU	GALC	GAL	QT	PT
ML	CU	OZFL	OZUK	TBSP	TSP
BBL	BU	PK	FBM		

TIME

YR	D	H	MIN	S	HZ

SPEED

M/S	CM/S	FT/S	KPH	MPH	KNOT
C	GA				

MASS

KG	G	LB	OZ	SLUG	LBT
TON	TONU	T	OZT	CT	GRAIN
U	MOL				

FORCE

N	DYN	GF	KIP	LBF	PDL

ENRG

J	ERG	KCAL	CAL	BTU	FT*LB
THER	MEV	EV			

POWR

W	HP				

PRESS

PA	ATM	BAR	PSI	TORR	MMH
INHG	INH20				

TEMP

°C	°F	K	°R		

ELEC

V	A	C	Ω	F	W
FDY	H	MHO	S	T	WB

ANGL

°	R	GRAD	ARCMI	ARCS	SR

LIGHT

FC	FLAM	LX	PH	SB	LM
CD	LAM				

RAD

GY	RAD	REM	SV	BQ	CI
R					

VISC

P	ST				

➡ UNITS

CONV	UBASE	UVAL	UFACT	→UNIT	

▲

ECHO	VIEW	PICK	ROLL	ROLLD	→LIST
DUPN	DRPN	KEEP	LEVEL		

D

UNITS and Interactive Stack Menus

E

HP 48 System Flags

This appendix lists the HP 48 system flags in functional groups. You can set, clear, and test all flags. The default state of the flags is *clear*—except for the Binary Integer Wordsize flags (flags −5 through −10).

System Flags

Flag	Description
	Symbolic Math Flags
−1	Principal Solution. *Clear*: QUAD and ISOL return a result representing all possible solutions. *Set*: QUAD and ISOL return only the principal solution.
−2	Symbolic Constants. *Clear*: Symbolic constants (e, i, π, MAXR, and MINR) retain their symbolic form when evaluated, unless the Numerical Results flag −3 is set. *Set*: Symbolic constants evaluate to numbers, regardless of the state of the Numerical Results flag −3.
−3	Numerical Results. *Clear*: Functions with symbolic arguments, including symbolic constants, evaluate to symbolic results. *Set*: Functions with symbolic arguments, including symbolic constants, evaluate to numbers.
−4	Not used.

System Flags (continued)

Flag	Description
	Binary Integer Math Flags
−5 thru −10	Binary Integer Wordsize. Combined states of flags −5 through −10 set the wordsize from 1 to 64 bits.
−11 and −12	Binary Integer Base. HEX: −11 *set*, −12 *set*. DEC: −11 *clear*, −12 *clear*. OCT: −11 *set*, −12 *clear*. BIN: −11 *clear*, −12 *set*.
−13 and −14	Not used.
	Coordinate System Flags
−15 and −16	Rectangular: −15 *clear*, −16 *clear*. Polar/Cylindrical: −15 *clear*, −16 *set*. Polar/Spherical: −15 *set*, −16 *set*.
	Trigonometric Angle Mode Flags
−17 and −18	Degrees: −17 *clear*, −18 *clear*. Radians: −17 *set*, −18 *clear*. Grads: −17 *clear*, −18 *set*.
	Complex Mode Flag
−19	*Clear*:→V2 and [→] [2D] create a 2-dimensional vector from 2 real numbers. *Set*:→V2 and [→] [2D] create a complex number from 2 real numbers.
	Math Exception-Handling Flags
−20	Underflow Exception. *Clear*: Underflow exception returns 0 and 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.

E

System Flags (continued)

Flag	Description
	Math Exception-Handling Flags (continued)
−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 thru −29	Not used.
	Plotting and Graphics Flags
−30	Function Plotting. *Clear*: For equations of form $y = f(x)$, only $f(x)$ is drawn. *Set*: For equations of form $y = f(x)$, separate plots of y and $f(x)$ are drawn.
−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.
	I/O and Printing Flags
−33	I/O Device. *Clear*: I/O directed to serial port. *Set*: I/O directed to IR port.
−34	Printing Device. *Clear*: Printer output directed to IR printer. *Set*: Printer output directed to serial port if flag −33 is clear.

E

System Flags (continued)

Flag	Description
	I/O and Printing Flags (continued)
−35	I/O Data Format. *Clear*: Objects transmitted in ASCII form. *Set*: Objects transmitted in binary (memory image) form.
−36	RECV Overwrite. *Clear*: If file name received by HP 48 matches existing HP 48 variable name, new variable name with number extension is created to prevent overwrite. *Set*: If file name received by HP 48 matches existing HP 48 variable name, existing variable is overwritten.
−37	Double-Spaced Printing. *Clear*: Single-spaced printing. *Set*: Double-spaced printing.
−38	Line Feed. *Clear*: Linefeed added at end of each print line. *Set*: No linefeed added at end of each print line.
−39	I/O Messages. *Clear*: I/O messages displayed. *Set*: I/O messages suppressed.
	Time Management Flags
−40	Clock Display. *Clear*: Ticking clock displayed only when TIME menu selected. *Set*: Ticking clock displayed at all times.
−41	Clock Format. *Clear*: 12-hour clock. *Set*: 24-hour clock.
−42	Date Format. *Clear*: MM/DD/YY (month/day/year) format. *Set*: DD.MM.YY (day.month.year) format.

System Flags (continued)

Flag	Description
Time Management Flags (continued)	
−43	Repeat Alarms Not Rescheduled. *Clear*: Unacknowledged repeat appointment alarms automatically rescheduled. *Set*: Unacknowledged repeat appointment alarms not rescheduled.
−44	Acknowledged Alarms Saved. *Clear*: Acknowledged appointment alarms deleted from alarm list. *Set*: Acknowledged appointment alarms saved in alarm list.
Display Format Flags	
−45 thru −48	Number of Decimal Digits. Combined states of flags −45 through −48 sets number of decimal digits in Fix, Scientific, and Engineering modes.
−49 and −50	Number Display Format. Standard: −49 *clear*, −50 *clear*. Fix: −49 *set*, −50 *clear*. Scientific: −49 *clear*, −50 *set*. Engineering: −49 *set*, −50 *set*.
−51	Fraction Mark. *Clear*: Fraction mark is . (period). *Set*: Fraction mark is , (comma).
−52	Single-Line Display. *Clear*: Display gives preference to object in level 1, using up to four lines of stack display. *Set*: Display of object in level 1 restricted to one line.
−53	Precedence. *Clear*: Certain parentheses in algebraic expressions suppressed to improve legibility. *Set*: All parentheses in algebraic expressions displayed.
−54	Not used.

System Flags (continued)

Flag	Description
	Miscellaneous Flags
−55	Last Arguments. *Clear*: Command arguments saved. *Set*: Command arguments not saved.
−56	Error Beep. *Clear*: Error and BEEP-command beeps enabled. *Set*: Error and BEEP-command beeps suppressed.
−57	Alarm Beep. *Clear*: Alarm beep enabled. *Set*: Alarm beep suppressed.
−58	Verbose Messages. *Clear*: Prompt messages and data automatically displayed. *Set*: Automatic display of prompt messages and data suppressed.
−59	Fast Catalog Display. *Clear*: Equation Catalog (and messages in SOLVE, SOLVR, PLOT, and PLOTR menus) show equation and equation name. *Set*: Equation Catalog (and messages in SOLVE, SOLVR, PLOT, and PLOTR menus) show equation name only.
−60	Alpha Lock. *Clear*: Single-Alpha activated by pressing [α] once. Alpha lock activated by pressing [α] twice. *Set*: Alpha lock activated by pressing [α] once. (Single-Alpha not available.)
−61	User-Mode Lock. *Clear*: 1-User mode activated by pressing [◂][USR] once. User mode activated by pressing [◂][USR] twice. *Set*: User mode activated by pressing [◂][USR] once. (1-User mode not available.)
−62	User Mode. *Clear*: User mode not active. *Set*: User mode active.

E

System Flags (continued)

Flag	Description
Miscellaneous Flags (continued)	
−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.

E

F

Comparing the HP 48 and HP 41

The HP 48 and the HP 41 share the "RPN" stack and "RPN" logic as the underlying basis of their operation. However, the four-level stack and fixed-register structure of the HP 41 aren't adequate for working with different types of data (different types of *objects*) and for doing symbolic calculations. The HP 48 has extended the traditional "RPN" model to provide the capabilities to carry out its enhanced operation.

To help you get started with the HP 48, this appendix highlights some of the similarities and differences between the HP 48 and HP 41. (If you're familiar with a different RPN calculator from HP, the comparisons in this appendix should still be helpful.) For a more extensive discussion, see *HP 41/HP 48 Transitions* by William C. Wickes, Larken Publications, 1990.

What's the Same

RPN Keyboard Calculations

You can calculate with numbers on the HP 48 just as you do on the HP 41. The underlying "RPN" operation is maintained.

The following examples calculate 4×15, sin 30°, and 2^3.

HP 41	HP 48
4 [ENTER↑] 15 [×]	4 [ENTER] 15 [×]

HP 41	HP 48
30 [SIN]	30 [SIN]

HP 41	HP 48
2 [ENTER] 3 ■ [y^x]	2 [ENTER] 3 [y^x]

Executing Commands

The general rule for the HP 41 and HP 48 is: *Enter the data for the command, then execute the command.* The results are normally returned to the stack.

The Stack

Stack Size

F

The HP 41 stack contains four registers (X, Y, Z, and T) augmented by the LAST X register and ALPHA register. Each of the numeric registers can hold one real-valued number (or six characters). The ALPHA register can hold 24 characters.

The HP 48 stack contains only as many *levels* as required for the number of objects entered. The stack starts at level 1 and grows as needed. (The display shows the first several levels.) Each stack level can contain one *object*—one item of data. An object can be a real number—or it can be a complex number, a string of characters of any length, a complete program, or one of several other types of data. The important point is that the stack is dynamic in both the number of levels and the complexity of information in each level.

If you want to enter text on the HP 48, you enter it as a string object. This means you can enter text on the stack—so there's no need for a separate ALPHA register.

Stack Memory

The HP 41 stack occupies a fixed portion of memory—its size never changes.

The dynamic HP 48 stack has the advantage that you never lose data off the stack as you enter new data. However, it has the disadvantage that you can tie up a significant amount of memory with old data if you leave it on the stack. You should get in the habit of discarding unneeded data from the stack.

Clearing the Stack

When you clear the HP 41 X-register or clear the stack, 0 is stored in the corresponding registers. You can use the 0's as data.

When you use CLEAR to clear the HP 48 stack, you delete all the stack objects and levels—there's *nothing* left in the stack—there's no data available there. If you use DROP to clear level 1 of the stack, all of the objects on the stack move down, leaving the former level 2 object in level 1 and one less stack level. No clearing operation generates a 0.

Entering Data

When you enter numbers on the HP 41, you enter them into the X-register. When you press [ENTER↑], the number is duplicated in the Y-register and stack-lift is disabled.

When you enter a number or other object on the HP 48, you enter it in the *command line*—so the stack levels aren't affected while you type. When you press [ENTER], the command line is processed—if you've entered a number, it's put in level 1 and the rest of the stack objects move up one level. If you press [ATTN] instead, the command line is deleted, and the stack isn't changed.

On the HP 48 you can actually enter more than one object in the command line by pressing [SPC] after each one. If you enter several numbers and press [ENTER], all of them are put on the stack in order, one number per level. If you press some other key, such as [+], it automatically processes the command line (the numbers go on the stack)—and then the command is executed.

Viewing the Stack

On the HP 41 you can use [R↓] to view the stack.

On the HP 48 you normally see up to four stack levels in the display. If you want to see more of the stack, you can press [▲] to activate the Interactive Stack, which lets you browse the entire stack.

Duplicating Stack Data

On the HP 41 you can press [ENTER↑] to make a copy of the X-register in the Y-register. And when you remove numbers from the stack, the number in the T-register is automatically duplicated in the Z-register—giving an infinite supply of T-register values.

On the HP 48 you execute DUP to duplicate the level 1 object in level 2. (For convenience, if there's no command line present, you can press [ENTER] to do the same thing.) You can use DUP2 and DUPN to duplicate two or more levels. However, there's no built-in HP 48 mechanism for automatic replication of data comparable to the HP 41 T-register operation—because the HP 48 stack doesn't have a fixed number of levels. (Of course, you can make a simple program that returns a given number each time you execute it.)

Reordering Stack Data

You can reorder the HP 41 stack data using R↑, RDN, and X<>Y.

F

The corresponding HP 48 commands are ROLL, ROLLD, and SWAP—*however*, ROLL and ROLLD require an argument that specifies how many stack levels to include in the "roll" operation. (The stack might have many levels containing data you don't want to affect.) Other stack commands are included in the PRG STK (stack) menu.

Calculations

Types of Calculations

HP 41 calculations use RPN syntax. (For a complex calculation, you usually start with the inner operations because of the fixed stack size.)

HP 48 calculations can use RPN syntax—called *stack* syntax. (You can calculate in any order because of the unlimited stack size.) Alternatively, you can calculate with *algebraic* syntax. Algebraic syntax specifies a calculation in algebraic notation—and you can preserve the calculation in this form or evaluate it for its numeric value.

These examples calculate $1/\sqrt{2}$. (The third example evaluates the expression '1/√2'.)

HP 41	HP 48	HP 48
1 [ENTER↑]	1 [ENTER]	['] 1 [÷] [√x] 2
2 [√x]	2 [√x]	[ENTER]
[÷]	[÷]	[EVAL]

Controlling the Display Format

The HP 41 provides three display formats: fixed, scientific, and engineering.

The HP 48 provides four display formats: fixed, scientific, engineering, and standard (the default). Standard format shows only as many digits as needed to represent the number (up to 12).

F

Rectangular and Polar Coordinates

On the HP 41 you use two registers to hold the rectangular or polar coordinates of a 2D vector or complex number. You use the R–P and P–R commands to convert between coordinate types.

On the HP 48 you use a 2D vector or complex number object to represent the quantity. You change the coordinate mode to change the way the object is displayed.

These examples show a 2D vector or complex number as polar coordinates.

HP 41	HP 48	HP 48
1 ENTER↑ 1 ■ R→P (x⇄y x⇄y to view)	1 ENTER 1 ↰ 2D ↱ POLAR	↰ ( ) 1 SPC 1 ENTER ↱ POLAR

Commands

Executing Commands

On the HP 41 you can press unshifted and shifted keys to execute keyboard commands. You can also press XEQ ALPHA and spell a command name to execute.

On the HP 48 commands appear on unshifted and shifted keys—notice the *left*-shift and *right*-shift prefix keys. Other commands are located in *menus*—you execute them by getting the menu and pressing the appropriate menu key (shown as ▒▒▒ in this manual). For example, you can press MTH PARTS ABS to execute the absolute-value command. You can also execute a command by typing its name in the command line and pressing ENTER—but you have to use α to type alpha characters.

These examples change from Radians mode to Degrees mode.

HP 41	HP 48	HP 48	HP 48
XEQ ALPHA DEG ALPHA	↰ RAD	↰ MODES NXT NXT DEG	α α DEG ENTER

No Prefix Notation

On the HP 41 certain commands require a register number, flag number, or other such parameter. Each such command prompts for the parameter *after* you execute the command name. For example, STO 01, FIX 2, and CF 03 get their parameters after you execute the command names.

On the HP 48 *all* commands require their arguments to be present on the stack *before* you execute the command. For example, you can execute 'N1' STO, 2 FIX, and 3 CF.

These examples change the display mode to show four decimal places.

HP 41	HP 48
■ [FIX] 4	4 [←] [MODES] FIX

Recovering Previous Data

On the HP 41 you can use LASTX to return the contents of the L-register—the previous contents of the X-register. This lets you reuse the data, perhaps to undo the last calculation.

On the HP 48 you can use LASTARG to return *all* of the arguments of the most recent command—not just the argument from level 1. You can use [LAST STACK] to restore the entire stack to the way it was before the last command.

Removing Input Data

For certain HP 41 commands, input data for the command is left on the stack after the command is executed. For example, 123 STO 01 leaves 123 on the stack.

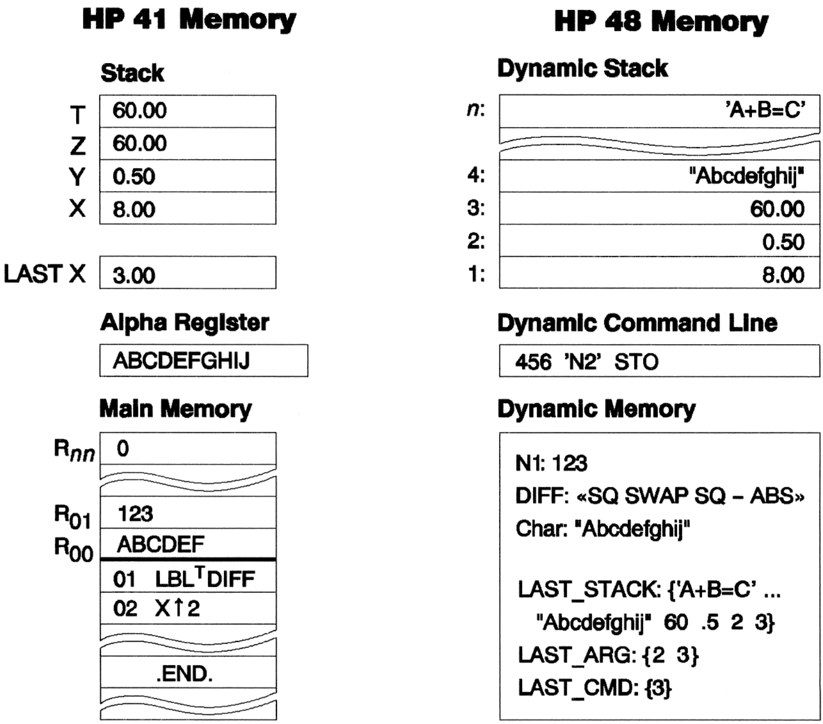
For almost all HP 48 commands, the input data for the command is removed from the stack. For example, 123 'N1' STO leaves nothing on the stack. This helps keep the stack uncluttered. If you want to keep a copy of an argument on the stack, you can use DUP to make an extra copy. For example, 123 DUP 'N1' STO leaves 123 on the stack. (If there's no command line, you can press [ENTER] to copy the level 1 object.)

Memory

Memory Organization

HP 41 main memory is separated into data storage registers and program memory. You set the amount of memory in each section using the SIZE command. Excess memory in either section isn't available for the opposite purpose. In addition, the HP 41 has a four-level stack and 24-character Alpha register.

HP 48 user memory is not divided or reserved for the purpose of storing different types of data, including objects on the stack. Memory is dynamically allocated for objects you enter or store—numbers, programs, or other types of objects. The number and size of objects is limited by only the amount of memory available.



Storing Data

On the HP 41 you store real numbers (and limited alpha data) in storage registers using the STO command. Each register is identified by a two-digit parameter, such as R_{01}. You have to remember what information is stored in what registers.

On the HP 48 you store real numbers, alpha strings of any length, complex numbers, and other types of objects in *variables*. Each variable is identified by a variable *name* that you give it—and it contains one object. The object can be a simple number, or a large program—or any other type of object. If you press the [VAR] key, you see the names of variables you've created. You can choose names that reflect the meanings of the data stored there, such as *N1*.

These examples store the number 123 (in register R_{01} or variable *N1*).

HP 41	HP 48	HP 48
123 [STO] 01	123 ['] [α] N1 [STO]	123 [VAR] [←] N1

Using Stored Data

On both the HP 41 and HP 48 you can recall and modify stored data. The HP 48 VAR menu provides shortcuts for working with variables.

These examples recall a stored number (from R_{01} or variable *N1*).

HP 41	HP 48	HP 48	HP 48
[RCL] 01	[VAR] N1	[α] N1 [ENTER]	['] [α] N1 [→] [RCL]

Clearing Memory

On the HP 41 you clear a storage register by storing 0 there. You can use CLRG to store 0 in all registers.

On the HP 48 you use PURGE to delete one or more variables—they're removed from memory and their space is recovered. You can use CLVAR to delete all variables.

F

Programming

Program Content

An HP 41 program consists of numbered lines with one command per line. A program typically begins with a LBL instruction and ends with an END or RTN instruction. Certain commands that take a parameter appear on one line, such as CF 03. The beginning label often serves as the "name" of the program.

An HP 48 program consists of a sequence of commands, numbers, and other objects enclosed between « » program quotes. There are no commands with attached parameters—all such arguments precede the command, such as 3 CF. A program object has no inherent "name"—but it gets a name when you store it in a variable, which you name.

These three programs take two numbers *x2* and *x1* from the stack and calculate $|x1^2 - x2^2|$. The keystrokes for entering the programs aren't shown. (The third program uses *local variables*, which aren't available on the HP 41.)

HP 41	**HP 48**	**HP 48**
[PRGM]	«	«
01 LBL⊤DIFF	SQ	→ x2 x1
02 X↑2	SWAP	'ABS(x1^2-x2^2)'
03 X<>Y	SQ	»
04 X↑2	-	['] [α] [α]
05 -	ABS	DIFF [α] [STO]
06 ABS	»	
07 END	['] [α] [α]	
[PRGM]	DIFF [α] [STO]	

Running Programs

On the HP 41 you can run a program by executing its name.

On the HP 48 you can run a program by typing its name or pressing its key in the VAR menu.

These examples enter the numbers 5 and 6 and run program *DIFF*.

HP 41	HP 48	HP 48
5 [ENTER] 6 [XEQ] [ALPHA] DIFF [ALPHA]	5 [ENTER] 6 [α] [α] DIFF [ENTER]	5 [ENTER] 6 [VAR] DIFF

Program Structure

An HP 41 program can have several entry points (LBL commands) and several exit points (GTO and RTN commands). Branching and looping are provided by conditional commands, GTO commands, and ISG and DSE commands. Subroutines are provided by XEQ and RTN commands—up to six pending returns are allowed.

An HP 48 program has only one entry point (the beginning) and only one exit point (the end). No "go-to" capability is available. However, powerful branching and looping *structures* are available, such as IF...THEN...ELSE...END and FOR...NEXT. Subroutines are provided by simply including the name of the "subroutine" program—there's no limit to the number of pending returns. All of these features support *structured programming* on the HP 48.

These program segments return the value 1 or 2 depending on the value of a number on the stack.

HP 41

```
26 X>0
27 GTO 00
28 2
29 GTO 01
30 LBL 00
31 1
32 LBL 01
```

HP 48

```
« ...
 IF 0 >
 THEN 1
 ELSE 2
 END
... »
```

HP 48

```
« ...
 → x
 'IFTE(x>0,1,2)'
... »
```

F

These program segments recall two numbers and execute subroutine *DIFF*.

HP 41	HP 48
14 RCL 08	« ...
15 RCL 09	X Y DIFF
16 XEQ ᵀDIFF	... »

These program segments calculate $\sum_{j=1}^{10} j^2$. (The first two segments use looping—the third uses the summation function.)

HP 41	HP 48	HP 48
32 1.010	« ...	« ...
33 STO 00	0	'Σ(j=1,10,SQ(j))'
34 CLX	1 10	EVAL
35 LBL 00	FOR j	... »
36 RCL 00	j SQ +	
37 INT	NEXT	
38 X^2	... »	
39 +		
40 ISG 00		
41 GTO 00		

Storing Programs

On the HP 41 programs are stored in program memory.

F

On the HP 48 you store programs in variables—just as you store other types of objects. You can also leave program objects on the stack—a program occupies just one stack level.

Managing Programs

On the HP 41 you usually manage your program memory—you pack program memory at certain times to maximize available space, and you use SIZE to change the amount of available program memory.

On the HP 48 memory is dynamically allocated—all of user memory is available for programs and other types of objects. The HP 48 automatically cleans up memory to maximize the available space.

Managing Intermediate Results

An HP 41 program must take into account the number of values on the stack. If more than four values are put on the stack, excess values are lost. You can use storage registers to compensate for the fixed stack size.

An HP 48 program can take as many arguments as needed from the stack—and it can return as many objects as needed to the stack—without concern for losing data. This lets you create building-block programs that perform given tasks and pass data to each other on the stack.

Using Flags

The HP 41 provides 56 flags. Flags 00 to 10 are general-purpose flags—flags 11 to 56 are system flags. You can change the settings of flags 00 to 29.

The HP 48 provides 128 flags (−1 to −64, and 1 to 64). Flags 1 to 64 are general-purpose flags—flags −1 to −64 are system flags. You can change all flag settings.

Labeling Output

An HP 41 program can label its output by using ARCL to combine a number with an alpha label. (The result is an alpha string, which can't be used directly in another calculation.)

An HP 48 program can show its output as an alpha string. Alternatively, it can "tag" a numeric result with an alpha label—and the result can still be used as a number in another calculation.

These program segments show the number 3 with the label "X". (The first two segments return alpha results—the third segment returns a numeric result, which you can use in further calculations.)

HP 41

```
53 FIX 2
54ᵀX=
55 3
56 ARCL X
57 AVIEW
```

HP 48

```
« ...
 2 FIX
 "X="
 3 →STR
 +
... »
```

HP 48

```
« ...
 2 FIX
 3 "X" →TAG
... »
```

F

G

Operation Index

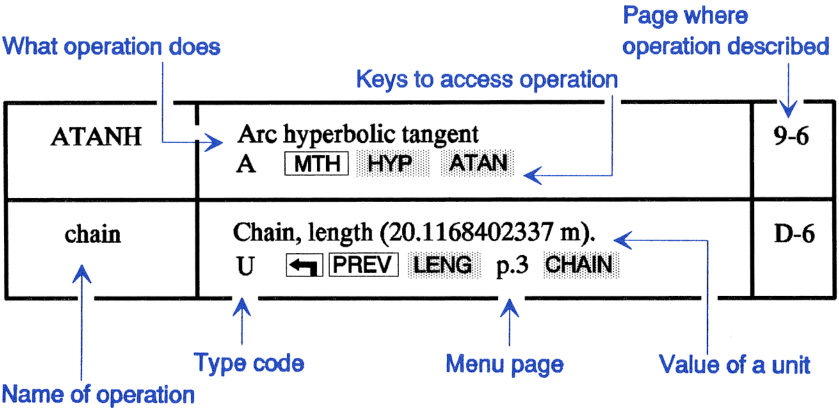
This index contains reference information for all operations in the HP 48. For each operation, this index shows:

- **Name, Key, or Label.** The name, key, or menu label associated with the operation. Operation names appear as keys or menu labels.
- **Description.** What the operation does (or its value if a unit).
- **Type.** The type of operation is given by one of the following codes:

Code	Description
O	**Operation.** An operation that cannot be included in the command line, in a program, or in an algebraic.
C	**Command.** An operation that can be included in programs but *not* in algebraics.
F	**Function.** A command that can be included in algebraics.
A	**Analytic Function.** A function for which the HP 48 provides an inverse and derivative.
U	**Unit.**

- **Keys.** The keys to access the operation. For keys preceded by " . . . ", you can access the operation through more than one menu—to see the keystrokes represented by the " . . . ", see the listing in this index for the operation that immediately follows the " . . . ". Operations in multipage menus show the applicable menu page number. Operations that aren't key-accessible are identified by "Must be typed in."
- **Page.** Where the operation is described in this manual.

The entries in this index are arranged as follows:



Operations whose names contain both alpha and special characters are listed alphabetically. Operations whose names contain special characters only are listed at the end of the index.

Name, Key, or Label	Description, Type, and Keys	Page
A→	Associate right. O ← EQUATION ◀ RULES →A	22-15
→ A→	Executes A→ until no change in subexpression. O ← EQUATION ◀ RULES → A→	22-18
ABS	Absolute value. MTH PARTS ABS MTH MATR p.2 ABS F MTH VECTR ABS	9-14
ACK	Acknowledges displayed past due alarm. C ← TIME ACK	24-7
ACKALL	Acknowledges all past due alarms. C ← TIME ACKA	24-7
ACOS	Arc cosine. A ← ACOS	9-9
ACOSH	Arc hyperbolic cosine. A MTH HYP ACOSH	9-6
acre	Acre, area (4046.87260987 m^2). U ← UNITS AREA p.2 ACRE	D-6
ADJST	Selects TIME ADJST (adjust) menu. O ← TIME ADJST	D-5
AF	Add fractions. O ← EQUATION ◀ RULES AF	22-17
← ALGEBRA	Selects ALGEBRA menu. O ← ALGEBRA	D-5
→ ALGEBRA	Selects Equation Catalog. O → ALGEBRA	17-7
ALOG	Common (base 10) antilogarithm. A ← 10^x	9-6
ALRM	Selects TIME ALRM (alarm) menu. O ← TIME ALRM	D-5

G

Name, Key, or Label	Description, Type, and Keys	Page
AND	Logical or binary AND. MTH BASE p.4 AND F PRG TEST AND	 14-5 26-3
ANGL	Selects UNITS ANGL menu. O ⬅ UNITS p.3 ANGL	D-6
APPLY	Returns evaluated expression(s) as argument(s) to unevaluated local name. F ⬅ ALGEBRA p.2 APPLY	
ARC	Draws arc in *PICT* from θ_1 to θ_2 with center at (x,y) and radius r. C PRG DSPL ARC	19-25
ARCHIVE	Makes backup copy of HOME directory. C ⬅ MEMORY p.3 ARCHI	34-18
arcmin	Minute of arc, plane angle. $(4.62962962963 \times 10^{-5})$ U ⬅ UNITS p.3 ANGL ARCMI	D-6
arcs	Second of arc, plane angle. $(7.71604938272 \times 10^{-7})$ U ⬅ UNITS p.3 ANGL ARCS	D-6
AREA	Calculates and displays area under function graph between two x-values specified by the mark and cursor; returns area to stack. O ... FCN AREA	18-26
AREA	Selects UNITS AREA menu. O ⬅ UNITS AREA	D-6
ARG	Returns polar angle θ. F MTH PARTS ARG	11-10
ARG	Enables/disables LASTARG recovery. O ⬅ MODES p.2 ARG	15-11
ARRY→	Returns array elements to stack. C Must be typed in.	

G

Name, Key, or Label	Description, Type, and Keys	Page
→ARRY	Combines numbers into array. C (PRG) OBJ →ARR	4-13
ASCII	Switches between ASCII and binary mode. O (↰)(I/O) SETUP ASCII	33-4
ASIN	Arc sine. A (↰)(ASIN)	9-9
ASINH	Arc hyperbolic sine. A (MTH) HYP ASINH	9-6
ASN	Makes a single user-key assignment. C (↱)(MODES) ASN	15-6
ASR	1-bit arithmetic shift right. C (MTH) BASE p.3 ASR	14-5
ATAN	Arc tangent. A (↰)(ATAN)	9-9
ATANH	Arc hyperbolic tangent. A (MTH) HYP ATAN	9-6
atm	Atmosphere, pressure (101325 kg/m·s^2) U (↰)(UNITS) p.2 PRESS ATM	D-6
ATTACH	Attaches specified library to current directory. C (↰)(MEMORY) p.2 ATTAC	34-21
(ATTN) ((ON))	Aborts program execution; aborts command line; exits special environments; clears messages. O (ON)	2-6
AU	Astronomical unit, length (1.495979 × 10^{11} m). U (↰)(UNITS) LENG p.2 AU	D-6
AUTO	Scales y-axis. ... PLOTR AUTO C (↱)(PLOT) AUTO	18-15

G

Name, Key, or Label	Description, Type, and Keys	Page
AUTO	Scales y-axis, then plots equation. ... PLOTR AUTO O (→)(PLOT) AUTO	18-15
AXES	Sets specified coordinates of axes intersection; stores labels. ... PLOTR p.3 AXES C (→)(PLOT) p.3 AXES	19-3
(→) AXES	Recalls axes intersection to stack. ... PLOTR p.3 (→) AXES O (→)(PLOT) p.3 (→) AXES	19-5
A/PM	Switches clock between AM and PM. O (←)(TIME) SET A/PM Switches alarm time between AM and PM. O (←)(TIME) ALRM A/PM	24-2 24-5
b	Barn, area (1×10^{-28} m^2). U (←)(UNITS) AREA B	D-6
bar	Bar, pressure (100000 $kg/m{\cdot}s^2$). U (←)(UNITS) p.2 PRESS BAR	D-6
BAR	Selects BAR plot type. C ... PTYPE BAR	19-13
BARPLOT	Draws bar plot of data in ΣDAT. C (←)(STAT) p.3 BARPL	21-19
BASE	Selects MTH BASE menu. O (MTH) BASE	D-3
BAUD	Sets one of four available baud rates. C (←)(I/O) SETUP BAUD	33-4
bbl	Barrel, volume (.158987294928 m^3). U (←)(UNITS) VOL p.4 BBL	D-6
BEEP	Sounds beep. C (PRG) CTRL p.3 BEEP	29-12

G

Name, Key, or Label	Description, Type, and Keys	Page
BEEP	Enables/disables error BEEP. O ⬅ MODES BEEP	15-11
BESTFIT	Selects statistics model yielding largest correlation coefficient (absolute value) and executes LR. C ⬅ STAT p.4 MODL BEST	21-11
BIN	Sets binary base. MTH BASE BIN C ⬅ MODES p.4 BIN	14-2
BINS	Sorts elements in independent variable column of ΣDAT into $N + 2$ bins (up to a maximum of 1048573 bins). C ⬅ STAT p.2 BINS	21-9
BLANK	Creates blank graphics object. C PRG DSPL p.3 BLAN	19-28
BOX	Draws box with opposite corners defined by specified coordinates. C PRG DSPL BOX	19-25
BOX	Draws box with opposite corners defined by mark and cursor. ... DRAW p.2 BOX ... AUTO p.2 BOX O ⬅ GRAPH p.2 BOX	19-23
Bq	Becquerel, activity (1 1/s). U ⬅ UNITS p.3 RAD BQ	D-6
BRCH	Selects PRG BRCH (program branch) menu. O PRG BRCH	D-3
Btu	International Table Btu, energy (1055.05585262 kg·m^2/s^2) U ⬅ UNITS p.2 ENRG BTU	D-6
bu	Bushel, volume (.03523907 m^3). U ⬅ UNITS VOL p.4 BU	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
BUFLEN	Returns number of characters in serial buffer. C ⬅ I/O p.3 BUFLE	33-21
BYTES	Returns object size (in bytes) and checksum for object. C ⬅ MEMORY BYTES	5-2
B→R	Binary-to-real conversion. C MTH BASE p.2 B→R	14-5
c	Speed of light (299792458 m/s). U ⬅ UNITS SPEED p.2 C	D-6
C	Coulomb, electric charge (1 A·s). U ⬅ UNITS p.2 ELEC C	D-6
°C	Degrees Celsius, temperature. U ⬅ UNITS p.2 TEMP °C	D-6
cal	Calorie, energy (4.186 kg·m^2/s^2) U ⬅ UNITS p.2 ENRG CAL	D-6
CASE	Begins CASE structure. C PRG BRCH CASE	26-6
⬅ CASE	Types CASE THEN END. O PRG BRCH ⬅ CASE	26-7
➡ CASE	Types THEN END. O PRG BRCH ➡ CASE	26-7
CAT	Selects Equation Catalog. ⬅ PLOT CAT ⬅ SOLVE CAT O ➡ ALGEBRA	17-7
	Selects STAT Catalog. O ⬅ STAT CAT	21-6
	Selects Alarm Catalog. ⬅ TIME CAT O ➡ TIME	24-12

G

Name, Key, or Label	Description, Type, and Keys	Page
cd	Candela, luminous intensity (1 cd). U (←)(UNITS) p.3 LIGHT p.2 CD	D-6
CEIL	Returns next greater integer. F (MTH) PARTS p.3 CEIL	9-14
CENT	Redraws graph with center at cursor position. ... DRAW CENT ... AUTO CENT O (←)(GRAPH) CENT	18-22
CENTR	Sets center of plot display at specified (x, y) coordinates. ... PLOTR p.2 CENT C (→)(PLOT) p.2 CENT	18-10
(→) CENT	Recalls plot-center coordinates to stack. ... PLOTR p.2 (→) CENT O (→)(PLOT) p.2 (→) CENT	18-16
CF	Clears specified flag. (PRG) TEST p.3 CF C (→)(MODES) p.2 CF	28-2
%CH	Returns % change from level 2 to level 1. F (MTH) PARTS p.2 %CH	9-7
chain	Chain, length (20.1168402337 m). U (←)(UNITS) LENG p.3 CHAIN	D-6
CHR	Converts character code to one-character string. C (PRG) OBJ p.3 CHR	4-13
Ci	Curie, activity (3.7×10^{10} 1/s). U (←)(UNITS) p.3 RAD CI	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
CIRCL	Draws circle with center at the mark and radius equal to the distance from cursor to mark. ... DRAW p.2 CIRCL ... AUTO p.2 CIRCL O (←)(GRAPH) p.2 CIRCL	19-23
CKSM	Selects one of three available checksum error-detect schemes. C (←)(I/O) SETUP CKSM	33-4
CLEAR	Clears stack. C (→)(CLR)	3-5
(CLR)	In EquationWriter entry mode, clears screen. O (←)(EQUATION) (→)(CLR) Clears *PICT*. ... DRAW (→)(CLR) ... AUTO (→)(CLR) O (←)(GRAPH) (→)(CLR)	16-4 18-21
CLK	Switches ticking clock display on and off. O (←)(MODES) p.2 CLK	15-11
CLKADJ	Adds specified number of clock ticks to system time. C (←)(TIME) ADJST p.2 CLKA	24-4
CLLCD	Blanks stack display. C (PRG) DSPL p.4 CLLCD	29-17
CLOSEIO	Closes I/O port. C (←)(I/O) p.2 CLOSE	33-18
CLΣ	Purges statistical data in *ΣDAT*. C (←)(STAT) CLΣ	21-2
CLUSR	Purges all user variables. C Must be typed in.	

G

Name, Key, or Label	Description, Type, and Keys	Page
CLVAR	Purges all user variables. C ⮫ PURGE	6-9
cm	Centimeter, length (.01 m). U ⮪ UNITS LENG CM	D-6
CMD	Enables/disables last command line recovery. O ⮪ MODES p.2 CMD	15-11
cm^2	Square centimeter, area (1×10^{-4} m^2). U ⮪ UNITS AREA CM^2	D-6
cm^3	Cubic centimeter, volume (1×10^{-6} m^3). U ⮪ UNITS VOL CM^3	D-6
cm/s	Centimeters per second, speed (.01 m/s). U ⮪ UNITS SPEED CM/S	D-6
CNCT	Switches curve filling on and off. O ⮪ MODES p.2 CNCT	18-14
CNRM	Calculates column norm of array. C MTH MATR p.2 CNRM	20-17
+COL	Inserts a row of zeros at current column in MatrixWriter application. O ⮫ MATRIX p.2 +COL	20-7
-COL	Deletes current column in MatrixWriter application. O ⮫ MATRIX p.2 -COL	20-7
COLCT	Collects like terms in expression. C ⮪ ALGEBRA COLCT	22-9
COLCT	Collects like terms in specified subexpression. O ⮪ EQUATION ◀ RULES COLCT	22-14
COLΣ	Specifies dependent and independent columns in ΣDAT. C Must be typed in.	

G

Name, Key, or Label	Description, Type, and Keys	Page
COMB	Returns number of combinations of n items taken m at a time. F (MTH) PROB COMB	9-13
CON	Creates constant array. C (MTH) MATR CON	20-17
CONIC	Selects CONIC plot type. C ... PTYPE CONIC	19-13
CONJ	Returns complex conjugate. F (MTH) PARTS CONJ	11-10
CONT	Continues halted program. C (◀)(CONT)	29-17
CONVERT	Converts unit object to dimensions of specified compatible unit. C (▶)(UNITS) CONV	13-9
COORD	Displays cursor coordinates at bottom left of display. ... DRAW COORD ... AUTO COORD O (◀)(GRAPH) COORD	18-20
CORR	Calculates correlation coefficient of statistical data in ΣDAT. C (◀)(STAT) p.4 CORR	21-12
COS	Cosine. A (COS)	9-9
COSH	Hyperbolic cosine. A (MTH) HYP COSH	9-6
COV	Calculates covariance of statistical data in ΣDAT. C (◀)(STAT) p.4 COV	21-12
CR	Causes printer to do carriage return/line feed. C (◀)(PRINT) CR	32-8

G

Name, Key, or Label	Description, Type, and Keys	Page
CRDIR	Creates a directory. C ⬑ MEMORY CRDIR	7-3
CROSS	Cross product of 2- or 3-element vector. C MTH VECTR CROSS	20-9
CST	Selects CST (custom) menu. O CST	15-2
CST	Returns contents of *CST* variable. O ⮫ MODES CST	15-2
ct	Carat, mass (.0002 kg). U ⬑ UNITS MASS p.2 CT	D-6
CTRL	Selects PRG CTRL (program control) menu. O PRG CTRL	D-3
cu	US cup, volume (2.365882365 × 10^{-4} m^3). U ⬑ UNITS VOL p.3 CU	D-6
C→PX	Converts user-unit coordinates to pixel coordinates. C PRG DSPL p.2 C→PX	19-9
C→R	Separates complex number into two real numbers. C PRG OBJ pg.2 C→R	4-13
d	Day, time (86400 s). U ⬑ UNITS TIME D	D-6
2D	Assembles or takes apart a complex number or 2D vector. O ⬑ 2D	12-4
3D	Assembles or takes apart a 3D vector. O ⮫ 3D	12-4

G

Name, Key, or Label	Description, Type, and Keys	Page
+D	Distribute left. O (←)(EQUATION) (◄) RULES +D	22-16
(→) +D	Executes +D until no change in subexpression. O (←)(EQUATION) (◄) RULES (→) +D	22-18
D→	Distribute right. O (←)(EQUATION) (◄) RULES D→	22-16
(→) D→	Executes D→ until no change in subexpression. O (←)(EQUATION) (◄) RULES (→) D→	22-18
DATE	Returns system date. C (←)(TIME) p.2 DATE	24-17
DATE+	Returns new date from specified date and number of days. C (←)(TIME) p.2 DATE+	24-17
→DATE	Sets specified system date. C (←)(TIME) SET →DAT	24-2
>DATE	Sets specified alarm date. O (←)(TIME) ALRM >DATE	24-5
DAY	Sets alarm repeat interval to n days. O (←)(TIME) ALRM RPT DAY	24-5
DBUG	Halts program execution before first object. O (PRG) CTRL DBUG	25-24
DDAYS	Returns number of days between two dates. C (←)(TIME) p.2 DDAYS	24-17
DEC	Sets decimal base. (MTH) BASE DEC C (←) MODES p.4 DEC	14-2
DECR	Decrements value of specified variable. C (→)(MEMORY) DECR	27-13

G

Name, Key, or Label	Description, Type, and Keys	Page
DEFINE	Creates variable or user-defined function. C ⬅ DEF	6-3 10-1
→DEF	Expands trigonometric and hyperbolic functions in terms of EXP and LN. O ⬅ EQUATION ◀ RULES →DEF	22-17
DEG	Sets Degrees mode. C ⬅ MODES p.3 DEG	9-8
DEL	Deletes character under cursor. O DEL	3-16
DEL	Erases area whose opposite corners are defined by mark and cursor. ... DRAW p.3 DEL ... AUTO p.3 DEL O ⬅ GRAPH p.3 DEL	19-23
←DEL	Deletes all characters from cursor to start of word. ⬅ EDIT ←DEL O ... EDIT ←DEL	3-8
➡ ←DEL	Deletes all characters from cursor to start of line. ⬅ EDIT ➡ ←DEL O ... EDIT ➡ ←DEL	3-8
DEL→	Deletes all characters from cursor to start of next word. ⬅ EDIT DEL→ O ... EDIT DEL→	3-8
➡ DEL→	Deletes all characters from cursor to end of line. ⬅ EDIT ➡ DEL→ O ... EDIT ➡ DEL→	3-8
DELALARM	Deletes alarm from system alarm list. C ⬅ TIME ALRM p.2 DELAL	24-16

Name, Key, or Label	Description, Type, and Keys	Page
DELAY	Sets delay time between lines sent to printer. C ⬅ PRINT p.2 DELAY	32-7
DELKEYS	Clears specified user-key assignment. C ➡ MODES DELK	15-8
DEPND	Specifies name of dependent plot variable. ... PLOTR p.2 DEPN C ➡ PLOT p.2 DEPN	19-2
➡ DEPN	Recalls dependent plot variable to stack. ... PLOTR p.2 ➡ DEPN O ➡ PLOT p.2 ➡ DEPN	19-5
DEPTH	Returns number of objects on stack. C PRG STK DEPTH	3-18
DET	Determinant of a matrix. C MTH MATR DET	20-17
DETACH	Detaches specified library from current directory. C ⬅ MEMORY p.2 DETAC	34-22
DINV	Double invert. O ⬅ EQUATION ◀ RULES DINV	22-13
DISP	Displays object in specified display line. PRG DSPL p.4 DISP C PRG CTRL p.2 DISP	29-4
DNEG	Double negate. O ⬅ EQUATION ◀ RULES DNEG	22-13
DO	Begins indefinite loop. C PRG BRCH DO	27-10
⬅ DO	Types DO UNTIL END. O PRG BRCH ⬅ DO	27-11
DOERR	Aborts program execution and displays specified message. C PRG CTRL p.3 DOERR	30-3

G

Name, Key, or Label	Description, Type, and Keys	Page
DOT	Dot product of two vectors. C (MTH) VECTR DOT	20-9
DOT+	Turns on pixels as cursor moves. ... DRAW p.2 DOT+ ... AUTO p.2 DOT+ O (←)(GRAPH) p.2 DOT+	19-23
DOT-	Turns off pixels as cursor moves. ... DRAW p.2 DOT- ... AUTO p.2 DOT- O (←)(GRAPH) p.2 DOT-	19-23
DRAW	Plots equation without axes. ... PLOTR DRAW C (→)(PLOT) DRAW	18-15
DRAW	Plots equation with axes. ... PLOTR DRAW O (→)(PLOT) DRAW	18-15
DRAX	Draws axes. ... PLOTR p.3 DRAX C (→)(PLOT) p.3 DRAX	19-6
DROP	Drops object in level 1; moves all remaining objects down one level. C (←)(DROP)	3-5
DROPN	Drops n objects from stack. C (PRG) STK pg.2 DRPN	3-18
DRPN	Drops all objects from stack at and below pointer. O ... ↑STK p.2 DRPN	3-11
DROP2	Drops first two objects from stack. C (PRG) STK pg.2 DROP2	3-18
DSPL	Selects PRG DSPL (program display) menu. O (PRG) DSPL	D-3

G

Name, Key, or Label	Description, Type, and Keys	Page
DTAG	Removes all tags from object. C (PRG) OBJ pg.2 DTAG	4-13
DUP	Duplicates object in level 1. C (PRG) STK pg.2 DUP	3-5
DUPN	Duplicates n objects on stack. C (PRG) STK pg.2 DUPN	3-19
DUPN	Duplicates all objects on stack from pointer through stack level 1. O ... ↑STK p.2 DUPN	3-11
DUP2	Duplicates objects in level 1 and level 2. C (PRG) STK pg.2 DUP2	3-19
dyn	Dyne, force (.00001 kg·m/s^2). U (↰)(UNITS) p.2 FORCE DYN	D-6
D→R	Degrees-to-radians conversion. F (MTH) VECTR p.2 D→R	9-11
e	Symbolic constant e (2.71828182846). F (α)(↰)(E)	9-15
ECHO	Copies object in current level to command line. O ... ↑STK ECHO	3-11
EDEQ	Returns contents of EQ to command line for editing. (↰)(PLOT) EDEQ O (↰)(SOLVE) EDEQ	17-11

G

Name, Key, or Label	Description, Type, and Keys	Page
EDIT	When command line not active, copies level-1 object into command line and selects EDIT menu.	3-7
	When command line active, selects EDIT menu.	3-8
	O ◄ EDIT	
	Selects EDIT menu.	20-7
	O ► MATRIX ◄ EDIT	
	Returns equation to command line and selects EDIT menu.	16-17
	O ◄ EQUATION ◄ EDIT	
	Edits current stack level.	3-12
	O ... ↑STK ◄ EDIT	
EDIT	Copies selected equation into command line and selects EDIT menu.	17-8
	◄ PLOT CAT EDIT	
	◄ SOLVE CAT EDIT	
	O ► ALGEBRA EDIT	
	Copies subexpression into command line and selects EDIT menu.	16-19
	O ◄ EQUATION ◄ EDIT	
	Copies selected matrix to MatrixWriter application.	21-7
	O ◄ STAT CAT EDIT	
	Edits current matrix cell.	20-7
	O ► MATRIX EDIT	
	Displays selected alarm and selects ALRM (alarm) menu.	24-12
	O ◄ TIME CAT EDIT	
EDITΣ	Copies statistical data in *ΣDAT* to MatrixWriter application.	21-4
	O ◄ STAT EDITΣ	

G

Name, Key, or Label	Description, Type, and Keys	Page
EEX	Types E or moves cursor to existing exponent in command line. O [EEX]	2-7
ELEC	Selects UNITS ELEC (electrical) menu. O [←][UNITS] p.2 ELEC	D-6
erg	Erg, energy (.0000001 kg·m^2/s^2) U [←][UNITS] p.2 ENRG ERG	D-6
ELSE	Begins false clause. C [PRG] BRCH p.3 ELSE	26-5
END	Ends program structures. C [PRG] BRCH p.2 END	26-5
ENG	Sets display mode to Engineering. C [←][MODES] ENG	2-15
ENRG	Selects UNITS ENRG (energy) menu. O [←][UNITS] p.2 ENRG	D-6
[ENTER]	Enters contents of command line. If no command line is present, executes DUP. O [ENTER]	3-2
[ENTRY]	Switches Algebraic- and Program-entry modes. O [→][ENTRY]	3-17
[EQUATION]	Selects EquationWriter application. O [←][EQUATION]	16-4
EQ+	Adds selected equation to list in *EQ*. [←][PLOT] CAT EQ+ [←][SOLVE] CAT EQ+ O [→][ALGEBRA] EQ+	17-27
[←] EQ+	Removes the last entry from the list in *EQ*. [←][PLOT] CAT [←] EQ+ [←][SOLVE] CAT [←] EQ+ O [→][ALGEBRA] [←] EQ+	17-27

G

Name, Key, or Label	Description, Type, and Keys	Page
EQ→	Separates equation into left and right sides. C (PRG) OBJ EQ→	4-13
ERASE	Erases *PICT*. ... PLOTR ERASE C (↱)(PLOT) ERASE	18-15
ERRM	Returns last error message. C (PRG) CTRL p.3 ERRM	30-3
ERRN	Returns last error number. C (PRG) CTRL p.3 ERRN	30-3
ERR0	Clears last error number. C (PRG) CTRL p.3 ERR0	30-3
eV	Electron volt, energy (1.60219×10^{-19} kg·m^2/s^2) U (↰)(UNITS) p.2 ENRG p.2 EV	D-6
EVAL	Evaluates object. C (EVAL)	4-20
EXEC	Sets alarm execution action. O (↰)(TIME) ALRM EXEC	24-11
(↱) EXEC	Recalls alarm execution action to stack. O (↰)(TIME) ALRM (↱) EXEC	24-11
EXECS	Shows alarm-execution action. (↰)(TIME) CAT EXECS O (↱)(TIME) EXECS	24-14
EXIT	Exits Selection environment. O (↰)(EQUATION) (◀) EXIT	22-12
	Exits FCN (function) menu. O ... FCN EXIT	18-26
	Exits ZOOM menu. O ... ZOOM EXIT	18-22

G

Name, Key, or Label	Description, Type, and Keys	Page
EXP	Constant e raised to power of object in level 1. A ⬑ e^x	9-6
EXPAN	Expands algebraic object. C ⬑ ALGEBRA EXPA	22-9
EXPFIT	Sets curve-fitting model to exponential. C ⬑ STAT p.4 MODL EXP	21-11
EXPM	Natural exponential minus 1 ($e^x - 1$). A MTH HYP p.2 EXPM	9-6
EXPR	Highlights subexpression for which specified object is top level function. O ⬑ EQUATION ◀ EXPR	16-22 22-11
EXPR=	Returns expression value or equation values. O ... SOLVR EXPR=	17-16
EXTR	Moves graphics cursor to nearest extremum, displays coordinates, and returns them to stack. O ... FCN EXTR	18-26
E^	Replace power-product with power-of-power. O ⬑ EQUATION ◀ RULES E^	22-17
E( )	Replace power-of-power with power-product. O ⬑ EQUATION ◀ RULES E( )	22-17
F	Farad, capacitance (1 $A^2 \cdot s^4/kg \cdot m^2$). U ⬑ UNITS p.2 ELEC F	D-6
°F	Degrees Fahrenheit, temperature. U ⬑ UNITS p.2 TEMP °F	D-6
FAST	Switches displaying equation names only and names plus contents of equations. O ... CAT p.2 FAST	17-8

G

Name, Key, or Label	Description, Type, and Keys	Page
fath	Fathom, length (1.82880365761 m). U ⬅ UNITS LENG p.3 FATH	D-6
fbm	Board foot, volume (.002359737216 m^3). U ⬅ UNITS VOL p.4 FBM	D-6
fc	Footcandle, illuminance (.856564774909 cd/m^2) U ⬅ UNITS p.3 LIGHT FC	D-6
FCN	Selects GRAPHICS FCN (function) menu. ... DRAW FCN ... AUTO FCN O ⬅ GRAPH FCN	D-4
	Plots statistical model. ⬅ STAT p.3 SCATR FCN	21-14
FC?	Tests if specified flag is clear. PRG TEST p.3 FC? C ➡ MODES p.3 FC?	28-2
FC?C	Tests if specified flag is clear, then clears it. PRG TEST p.3 FC?C C ➡ MODES p.3 FC?C	28-2
Fdy	Faraday, electric charge (96487 A·s). U ⬅ UNITS p.2 ELEC p.2 FDY	D-6
fermi	Fermi, length (1×10^{-15} m). U ⬅ UNITS LENG p.4 FERMI	D-6
FINDALARM	Returns first alarm due after specified time. C ⬅ TIME ALRM p.2 FINDA	24-16
FINISH	Terminates Kermit server mode. C ⬅ I/O FINIS	33-18
FIX	Selects Fix display mode. C ⬅ MODES FIX	2-15

G

Name, Key, or Label	Description, Type, and Keys	Page
flam	Footlambert, luminance ($3.42625909964\ cd/m^2$) U (←)(UNITS) p.3 LIGHT FLAM	D-6
FLOOR	Next smaller integer. F (MTH) PARTS p.3 FLOOR	9-14
FM,	Switches period and comma fraction mark. O (←) MODES p.4 FM,	2-15
FOR	Begins definite loop. C (PRG) BRCH FOR	27-6
(←) FOR	Types FOR NEXT. O (PRG) BRCH (←) FOR	27-7
(→) FOR	Types FOR STEP. O (PRG) BRCH (→) FOR	27-9
FORCE	Selects UNITS FORCE menu. O (←)(UNITS) p.2 FORCE	D-6
FP	Returns fractional part of a number. F (MTH) PARTS p.3 FP	9-14
FREE	Frees the memory in a previously merged RAM card. C (←)(MEMORY) p.3 FREE	34-12
FREEZE	Freezes one or more of three display areas. C (PRG) DSPL p.4 FREEZ	19-29 29-4
FS?	Tests if specified flag is set. (PRG) TEST p.3 FS? C (→)(MODES) p.3 FS?	28-2
FS?C	Tests if specified flag is set, then clears it. (PRG) TEST p.3 FS?C C (→)(MODES) p.3 FS?C	28-2
ft	International foot, length (.3048 m). U (←)(UNITS) LENG FT	D-6
ft^2	Square foot, area ($.09290304\ m^2$). U (←)(UNITS) AREA FT^2	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
ft^3	Cubic foot, volume (.028316846592 m^3). U ⬅ UNITS VOL FT^3	D-6
ftUS	Survey foot, length (.304800609601 m). U ⬅ UNITS LENG p.3 FTUS	D-6
ft/s	Feet/second, speed (.3048 m/s). U ⬅ UNITS SPEED FT/S	D-6
ft*lbf	Foot-poundf, energy (1.35581794833 kg·m^2/s^2). U ⬅ UNITS p.2 ENRG FT*LB	D-6
FUNCTION	Selects FUNCTION plot type. C ... PTYPE FUNC	19-13
F(X)	Displays value of function at x-value specified by cursor. Returns function value to stack. O ... FCN p.2 F(X)	18-27
F'	Plots first derivative of function, replots function, and adds derivative to *EQ*. O ... FCN p.2 F'	18-27
g	Gram, mass (.001 kg). U ⬅ UNITS MASS G	D-6
ga	Standard freefall, acceleration (9.80665 m/s^2). U ⬅ UNITS SPEED p.2 GA	D-6
gal	US gallon, volume (.003785411784 m^3). U ⬅ UNITS VOL p.2 GAL	D-6
galC	Canadian gallon, volume (.00454609m^3). U ⬅ UNITS VOL p.2 GALC	D-6
galUK	UK gallon, volume (.004546092 m^3). U ⬅ UNITS VOL p.2 GALU	D-6
GET	Gets element from array or list. C PRG OBJ p.4 GET	4-14

G

Name, Key, or Label	Description, Type, and Keys	Page
GETI	Gets element from array or list and increments index. C (PRG) OBJ p.4 GETI	4-14
gf	Gram-force (.00980665 kg·m/s^2). U (←)(UNITS) p.2 FORCE GF	D-6
GOR	Superposes graphics object onto graphics object. C (PRG) DSPL p.3 GOR	19-28
GO↓	Sets top-to-bottom entry mode. O (→)(MATRIX) GO↓	20-7
GO→	Sets left-to-right entry mode. O (→)(MATRIX) GO→	20-7
GRAD	Selects Grads mode. C (←)(MODES) p.3 GRAD	9-8
grad	Grade, plane angle (.0025). U (←)(UNITS) p.3 ANGL GRAD	D-6
grain	Grain, mass (.00006479891 kg). U (←)(UNITS) MASS p.2 GRAIN	D-6
GRAPH	Enters Graphics environment. C (←)(GRAPH)	18-19
(GRAPH)	Invokes scrolling mode. (←)(EQUATION) (←)(GRAPH) ... DRAW (←)(GRAPH) ... AUTO (←)(GRAPH) O (←)(GRAPH) (←)(GRAPH)	 16-3 18-20
→GROB	Converts object into graphics object. C (PRG) DSPL p.3 →GRO	19-28
GXOR	Superposes inverting graphics object onto graphics object. C (PRG) DSPL p.3 GXOR	19-28
Gy	Gray, absorbed dose (1 m^2/s^2). U (←)(UNITS) p.3 RAD GY	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
h	Hour, time (3600 s). U ⬅ UNITS TIME H	D-6
H	Henry, inductance (1 kg·m^2/A^2·s^2). U ⬅ UNITS p.2 ELEC p.2 H	D-6
*H	Adjusts vertical plot scale. C ... PLOTR p.3 *H	19-6
ha	Hectare, area (10000 m^2). U ⬅ UNITS AREA p.2 HA	D-6
HALT	Halts program execution. C PRG CTRL HALT	25-24 29-4
HEX	Sets hexadecimal base. MTH BASE HEX C ⬅ MODES p.4 HEX	14-2
HISTPLOT	Draws histogram of data in *ΣDAT*. C ⬅ STAT p.3 HISTP	21-19
HISTOGRAM	Selects HISTOGRAM plot type. C ... PTYPE p.2 HIST	19-13
HMS+	Adds in HMS format. C ⬅ TIME p.3 HMS+	9-11 24-19
HMS−	Subtracts in HMS format. C ⬅ TIME p.3 HMS−	9-11 24-19
HMS→	Converts from HMS to decimal format. C ⬅ TIME p.3 HMS→	9-11 24-19
→HMS	Converts base 10 number to HMS format. C ⬅ TIME p.3 →HMS	9-11 24-19
HOME	Selects *HOME* directory. C ➡ HOME	7-5
HOUR	Sets alarm repeat interval to *n* hours. O ⬅ TIME ALRM RPT HOUR	24-5
hp	Horsepower, power (745.699871582 kg·m^2/s^3). U ⬅ UNITS p.2 POWR HP	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
HR+	Increments time by one hour. O ⬅ TIME ADJUST HR+	24-4
HR-	Decrements time by one hour. O ⬅ TIME ADJUST HR-	24-4
HYP	Selects MTH HYP (math hyperbolic) menu. O MTH HYP	D-3
Hz	Hertz, frequency (1/s). U ⬅ UNITS TIME HZ	D-6
i	Symbolic constant *i*. F α ⬅ CST	9-15
IDN	Creates identity matrix of specified size. C MTH MATR IDN	20-17
IF	Begins test clause. C PRG BRCH IF	26-5
⬅ IF	Types IF THEN END. O PRG BRCH ⬅ IF	26-5
➡ IF	Types IF THEN ELSE END. O PRG BRCH ➡ IF	26-6
IFERR	Begins test clause. C PRG BRCH p.3 IFERR	30-4
⬅ IFERR	Types IFERR THEN END. O PRG BRCH p.3 ⬅ IFERR	30-4
➡ IFERR	Types IFERR THEN ELSE END. O PRG BRCH p.3 ➡ IFERR	30-5
IFT	IF-THEN command. C PRG BRCH p.3 IFT	26-5
IFTE	IF-THEN-ELSE function. F PRG BRCH p.3 IFTE	26-6
IM	Returns imaginary part of complex number or array. F MTH PARTS IM	11-10

Name, Key, or Label	Description, Type, and Keys	Page
in	Inch, length (.0254 m). U ⬅ UNITS LENG IN	D-6
in^2	Square inch, area (.00064516 m^2). U ⬅ UNITS AREA IN^2	D-6
in^3	Cubic inch, volume (.000016387064 m^3). U ⬅ UNITS VOL IN^3	D-6
INCR	Increments value of specified variable. C ➡ MEMORY INCR	27-13
INDEP	Specifies independent variable in a plot. ... PLOTR INDEP C ➡ PLOT INDEP	18-16
➡ INDEP	Recalls independent variable to stack. ... PLOTR ➡ INDEP O ➡ PLOT ➡ INDEP	18-16
inHg	Inches of mercury, pressure (3386.38815789 kg/m·s^2). U ⬅ UNITS p.2 PRESS p.2 INHG	D-6
inH2O	Inches of water, pressure (248.84 kg/m·s^2). U ⬅ UNITS p.2 PRESS p.2 INH2O	D-6
INPUT	Suspends program execution, displays message, and waits for data. C PRG CTRL p.2 INPUT	29-5
INS	Switches between insert/replace character. O ⬅ EDIT INS	3-8
INV	Reciprocal (inverse). A 1/x	9-3
⬅ I/O	Selects I/O (input/output) menu. O ⬅ I/O	D-4
➡ I/O	Selects Kermit server. O ➡ I/O	33-18

G

Name, Key, or Label	Description, Type, and Keys	Page
IP	Integer part of real number. F (MTH) PARTS p.3 IP	9-14
IR/W	Switches IR and Wire transmission modes. O (←)(I/O) SETUP IR/W	33-4
ISECT	Moves graphics cursor to closest intersection in two-function plot, displays intersection coordinates, and returns coordinates to stack. O ... FCN ISECT	18-26
ISOL	Isolates variable on one side of equation. C (←)(ALGEBRA) ISOL	22-2
J	Joule, energy (1 kg·m^2/s^2). U (←)(UNITS) p.2 ENRG J	D-6
K	Kelvins, temperature (1 K). U (←)(UNITS) p.2 TEMP K	D-6
kcal	Kilocalorie, energy (4186 kg·m^2/s^2) U (←)(UNITS) p.2 ENRG KCAL	D-6
KEEP	Clears all levels above current level. O ... ↑STK p.2 KEEP	3-12
KERRM	Returns text of most recently-received KERMIT error packet. C (←)(I/O) p.2 KERR	33-18
KEY	Returns number indicating last key pressed. C (PRG) CTRL p.2 KEY	29-14
KEYS	Removes menu labels. ... DRAW p.3 KEYS ... AUTO p.3 KEYS O (←)(GRAPH) p.3 KEYS	18-20
kg	Kilogram, mass (1 kg). U (←)(UNITS) MASS KG	D-6

Name, Key, or Label	Description, Type, and Keys	Page
KGET	Gets data from another device. C ⬏ I/O KGET	33-18
KILL	Aborts all suspended programs. C PRG CTRL KILL	25-24
kip	Kilopound-force (4448.22161526 kg·m/s^2). U ⬏ UNITS p.2 FORCE KIP	D-6
km	Kilometer, length (1 km). U ⬏ UNITS LENG p.2 KM	D-6
km^2	Square kilometer, area (1 km^2). U ⬏ UNITS AREA p.2 KM^2	D-6
knot	Nautical miles per hour, speed (.514444444444 m/s). U ⬏ UNITS SPEED KNOT	D-6
kph	Kilometers per hour, speed (.277777777778 m/s). U ⬏ UNITS SPEED KPH	D-6
l	Liter, volume (.001 m^3). U ⬏ UNITS VOL p.2 L	D-6
LABEL	Labels axes with variable names and ranges. ... PLOTR p.3 LABEL C ⬎ PLOT p.3 LABEL	19-3
LABEL	Labels axes with variable names and ranges. ... DRAW LABEL ... AUTO LABEL O ⬏ GRAPH LABEL	18-20
lam	Lambert, luminance (3183.09886184 cd/m^2). U ⬏ UNITS p.3 LIGHT p.2 LAM	D-6
LAST	Returns previous argument(s) to stack. C Must be keyed in.	

Name, Key, or Label	Description, Type, and Keys	Page
LASTARG	Returns previous argument(s) to stack. C ⮫ LAST ARG	3-5
LAST CMD	Displays previous contents of command line. O ⮪ LAST CMD	3-18
LAST MENU	Selects last displayed page of previous menu. O ⮫ LAST MENU	2-13
LAST STACK	Restores previous stack. O ⮪ LAST STACK	3-6
lb	Avoirdupois pound, mass (.45359237 kg). U ⮪ UNITS MASS LB	D-6
lbf	Pound-force (4.44822161526 kg·m/s^2). U ⮪ UNITS p.2 FORCE LBF	D-6
lbt	Troy pound, mass (.3732417 kg). U ⮪ UNITS MASS LBT	D-6
LCD→	Returns graphics object to stack representing stack display. C PRG DSPL p.4 LCD→	19-29
→LCD	Displays specified graphics object in stack display. C PRG DSPL p.4 →LCD	19-29
LENG	Selects UNITS LENG (length) menu. O ⮪ UNITS LENG	D-6
LEVEL	Enters current level number into level 1. O ... ↑STK p.2 LEVEL	3-12
LIBRARY	Selects LIBRARY menu. O ⮪ LIBRARY	34-16 34-22
LIBS	Lists all libraries attached to current directory. C ⮪ MEMORY p.2 LIBS	34-23

G

Name, Key, or Label	Description, Type, and Keys	Page
LIGHT	Selects UNITS LIGHT menu. O ⬅ UNITS p.3 LIGHT	D-6
LINE	Draws line between coordinates in levels 1 and 2. C PRG DSPL LINE	19-25
LINE	Draws line from mark to cursor. ... DRAW p.2 LINE ... AUTO p.2 LINE O ⬅ GRAPH p.2 LINE	19-23
ΣLINE	Returns best-fit line for data in *ΣDAT* according to selected statistical model. C ⬅ STAT p.3 ΣLINE	21-11
LINFIT	Sets curve-fitting model to linear. C ⬅ STAT p.4 MODL LIN	21-11
LIST→	Returns list elements to stack. C Must be typed in.	
→LIST	Combines specified objects into list. C PRG OBJ →LIST	4-14
→LIST	Combines objects from level 1 to current level into a list. O ... ↑STK →LIST	3-11
lm	Lumen, luminous flux ($7.95774715459 \times 10^{-2}$ cd). U ⬅ UNITS p.3 LIGHT LM	D-6
LN	Natural (base e) logarithm. A ➡ LN	9-6
LNP1	Natural logarithm of (*argument* + 1). A MTH HYP p.2 LNP1	9-6
LOG	Common (base 10) logarithm. A ➡ LOG	9-6
LOGFIT	Set curve-fitting model to logarithmic. C ⬅ STAT p.4 MODL LOG	21-11

G

Name, Key, or Label	Description, Type, and Keys	Page
LR	Calculates linear regression. C (◄)(STAT) p.4 LR	21-12
lx	Lux, illuminance ($7.95774715459 \times 10^{-2}$ cd/m^2). U (◄)(UNITS) p.3 LIGHT LX	D-6
lyr	Light year, length ($9.46052840488 \times 10^{15}$ m). U (◄)(UNITS) LENG p.2 LYR	D-6
L*	Replace log-of-power with product-of-log. O (◄)(EQUATION) (◄) RULES L*	22-17
L()	Replace product-of-log with log-of-power. O (◄)(EQUATION) (◄) RULES L()	22-17
←M	Merge-factors-left. O (◄)(EQUATION) (◄) RULES ←M	22-16
(➦) ←M	Executes ←M until no change in subexpression. O (◄)(EQUATION) (◄) RULES (➦) ←M	22-18
M→	Merge-factors-right. O (◄)(EQUATION) (◄) RULES M→	22-16
(➦) M→	Executes M→ until no change in subexpression. O (◄)(EQUATION) (◄) RULES (➦) M→	22-18
m	Meter, length (1 m). U (◄)(UNITS) LENG M	D-6
m^2	Square meter, area (1 m^2). U (◄)(UNITS) AREA M^2	D-6
m^3	Cubic meter (Stere), volume (1 m^3). U (◄)(UNITS) VOL M^3	D-6
MANT	Mantissa (decimal part) of number. F (MTH) PARTS p.3 MANT	9-14

Name, Key, or Label	Description, Type, and Keys	Page
MARK	Sets mark at cursor position. ... DRAW p.3 MARK ... AUTO p.3 MARK O ⬑ GRAPH p.3 MARK	18-20
MASS	Selects UNITS MASS menu. O ⬑ UNITS MASS	D-6
↑MATCH	Match-and-replace, beginning with subexpressions. C ⬑ ALGEBRA p.2 ↑MAT	22-24
↓MATCH	Match-and-replace, beginning with top-level expression. C ⬑ ALGEBRA p.2 ↓MAT	22-24
MATR	Selects MTH MATR (math matrices) menu. O MTH MATR	D-3
MATRIX	Selects MatrixWriter application. O ⬏ MATRIX	20-3
MAX	Maximum of two real numbers. F MTH PARTS p.2 MAX	9-14
MAXR	Maximum machine-representable real number (9.99999999999E499). F MTH PARTS p.4 MAXR	9-15
MAXΣ	Maximum column values in statistics matrix in ΣDAT. C ⬑ STAT p.2 MAXΣ	21-9
MEAN	Calculates mean of statistical data in ΣDAT. C ⬑ STAT p.2 MEAN	21-9
MEM	Bytes of available memory. C ⬑ MEMORY MEM	5-2

G

Name, Key, or Label	Description, Type, and Keys	Page
⬅ MEMORY	Selects MEMORY menu. O ⬅ MEMORY	D-4
➡ MEMORY	Selects MEMORY Arithmetic menu. O ➡ MEMORY	D-4
MENU	Displays built-in or custom menu. ➡ MODES MENU C PRG CTRL p.2 MENU	 15-1 29-18
MERGE	Merges plug-in RAM card memory with main memory. C ⬅ MEMORY p.3 MERG	34-11
μ	Micron, length (1×10^{-6} m). U ⬅ UNITS LENG p.4 μ	D-6
MeV	Mega electron volt, energy (1.60219×10^{-13} kg·m^2/s^2). U ⬅ UNITS p.2 ENRG p.2 MEV	D-6
mho	Mho, electric conductance (1 A^2·s^3/kg·m^2). U ⬅ UNITS p.2 ELEC p.2 MHO	D-6
mi	International mile, length (1609.344 m). U ⬅ UNITS LENG p.2 MI	D-6
mi^2	International square mile, area (2589988.11034 m^2). U ⬅ UNITS AREA p.2 MI^2	D-6
mil	Mil, length (.0000254 m). U ⬅ UNITS LENG p.4 MIL	D-6
min	Minute, time (60 s). U ⬅ UNITS TIME MIN	D-6
MIN	Minimum of two real numbers. F MTH PARTS p.2 MIN	9-14
MIN	Sets alarm repeat interval in minutes. O ⬅ TIME ALRM RPT MIN	24-5

Name, Key, or Label	Description, Type, and Keys	Page
MINR	Minimum machine-representable real number (1.00000000000E−499). F (MTH) PARTS p.4 MINR	9-15
MIN+	Increments system time by one minute. O (⬑)(TIME) ADJST MIN+	24-4
MIN-	Decrements system time by one minute. O (⬑)(TIME) ADJST MIN-	24-4
MINΣ	Finds minimum column values in statistics matrix in *ΣDAT*. C (⬑)(STAT) p.2 MINΣ	21-9
miUS	US statute mile, length (1609.34721869 m). U (⬑)(UNITS) LENG p.3 MIUS	D-6
miUS^2	US statute square mile, area (258998.47032 m^2). U (⬑)(UNITS) AREA p.2 MIUS^	D-6
mm	Millimeter, length (.001 m). U (⬑)(UNITS) LENG MM	D-6
mmHg	Millimeter of mercury (torr), pressure (133.322368421 $kg/m{\cdot}s^2$). U (⬑)(UNITS) p.2 PRESS MMH	D-6
ml	Milliliter (cubic centimeter), volume (1×10^{-6} m^3). U (⬑)(UNITS) VOL p.3 ML	D-6
ML	Switches multi-line and single-line display. O (⬑)(MODES) p.2 ML	15-11
MOD	Modulo. F (MTH) PARTS p.2 MOD	9-14
(⬑)(MODES) (⬏)(MODES)	Selects MODES menu. O (⬑)(MODES) Selects MODES Customization menu. O (⬏)(MODES)	D-4 D-4

G

Name, Key, or Label	Description, Type, and Keys	Page
MODL	Selects STAT MODL (statistics model) menu. O (←)(STAT) p.4 MODL	21-11
mol	Mole, mass (1 mol). U (←)(UNITS) MASS p.3 MOL	D-6
Mpc	Megaparsec, length ($3.08567818585 \times 10^{22}$ m). U (←)(UNITS) LENG p.2 MPC	D-6
mph	Miles per hour, speed (.44704 m/s). U (←)(UNITS) SPEED MPH	D-6
(MTH)	Selects MTH (math) menu. O (MTH)	D-3
M/D	Switches date display format. O (←)(TIME) SET M/D	24-2
m/s	Meters per second, speed (1 m/s). U (←)(UNITS) SPEED M/S	D-6
N	Newton, force (1 kg·m/s^2). U (←)(UNITS) p.2 FORCE N	D-6
NΣ	Returns number of rows in *ΣDAT*. C (←)(STAT) p.5 NΣ	21-20
NEG	Negate. A (+/-)	9-3
NEW	Takes algebraic or matrix from stack, prompts for name, stores named algebraic in *EQ*, or named matrix in *ΣDAT*. (←)(PLOT) NEW (←)(SOLVE) NEW O (←)(STAT) NEW	 18-5 17-4 21-3
NEWOB	Converts object taken from a composite object or variable into a new, independent object. C (←)(MEMORY) p.2 NEWO	

G

Name, Key, or Label	Description, Type, and Keys	Page
NEXT	Ends a definite-loop structure. C (PRG) BRCH p.2 NEXT	27-2 27-6
NEXT	Displays but does not execute next one or two objects in suspended program. O (PRG) CTRL NEXT	25-24
nmi	Nautical mile, length (1852 m). U (↰)(UNITS) LENG p.3 NMI	D-6
NONE	Cancels alarm repeat interval and returns to TIME ALRM menu. O (↰)(TIME) ALRM RPT NONE	24-5
NOT	Logical or binary NOT. (PRG) TEST NOT F (MTH) BASE p.4 NOT	 26-3 14-5
NUM	Returns character code of first character in string. C (PRG) OBJ p.3 NUM	4-14
→NUM	Evaluates algebraic to number. C (↱)(→NUM)	8-4
NXEQ	Rotates list of equations in *EQ*. ... SOLVR NXEQ O ... FCN p.2 NXEQ	 17-27 18-27
(NXT)	Selects next page of menu. O (NXT)	2-12
OBJ	Selects PRG OBJ (program object) menu. O (PRG) OBJ	D-3
OBJ→	Returns object components to stack. C (PRG) OBJ OBJ→	4-15
OCT	Sets octal base. (MTH) BASE OCT C (↰) MODES p.4 OCT	14-2
(OFF)	Turns calculator off. O (↱)(OFF)	1-4

G

Name, Key, or Label	Description, Type, and Keys	Page
OFF	Turns calculator off. C (PRG) CTRL p.3 OFF	29-23
OLDPRT	Remaps HP 48 character set to match HP 82240A Infrared Printer. C (←)(PRINT) p.2 OLDPR	32-2
(ON)	Turns calculator on. O (ON)	1-4
OPENIO	Opens serial port. C (←)(I/O) p.2 OPENI	33-18
OR	Logical or binary OR. (MTH) BASE p.4 OR F (PRG) TEST OR	 14-5 26-3
ORDER	Rearranges VAR menu in order specified in list. C (←)(MEMORY) ORDER	6-8
ORDER	Puts selected equation at top of Equation Catalog list. (←)(PLOT) CAT p.2 ORDER (←)(SOLVE) CAT p.2 ORDER O (→)(ALGEBRA) p.2 ORDER Puts selected statistical data at top of Statistics Catalog list. O (←)(STAT) CAT p.2 ORDER	17-8 21-7
OVER	Duplicates object in level 2 in level 1. C (PRG) STK OVER	3-19
oz	Ounce, mass (.028349523135 kg). U (←)(UNITS) MASS OZ	D-6
ozfl	US fluid ounce, volume ($2.95735295625 \times 10^{-5}$ m^3). U (←)(UNITS) VOL p.3 OZFL	D-6
ozt	Troy ounce, mass (.031103475 kg). U (←)(UNITS) MASS p.2 OZT	D-6

Name, Key, or Label	Description, Type, and Keys	Page
ozUK	UK fluid ounce, volume (2.8413075×10^{-5} m^3). U ⬅ UNITS VOL p.3 OZUK	D-6
P	Poise, dynamic viscosity (.1 kg/m·s) U ⬅ UNITS p.3 VISC P	D-6
Pa	Pascal, pressure (1 kg/m·s^2) U ⬅ UNITS p.2 PRESS PA	D-6
PARAMETRIC	Selects PARAMETRIC plot type. C ... PTYPE PARA	19-13
PARITY	Selects one of 5 possible parity settings. C ⬅ I/O SETUP PARIT	33-4
PARTS	Selects MTH PARTS menu. O MTH PARTS	D-3
PATH	Returns list containing path to current directory. C ⬅ MEMORY PATH	7-3
pc	Parsec, length ($3.08567818585 \times 10^{16}$ m). U ⬅ UNITS LENG p.2 PC	D-6
PDIM ➡ PDIM	Changes size of *PICT*. ... PLOTR p.3 PDIM C ➡ PLOT p.3 PDIM Recalls size of *PICT* to stack. ... PLOTR p.3 ➡ PDIM O ➡ PLOT p.3 ➡ PDIM	19-9 19-6
pdl	Poundal, force (.138254954376 kg·m/s^2). U ⬅ UNITS p.2 FORCE PDL	D-6
PERM	Permutations. F MTH PROB PERM	9-13
PGDIR	Purges specified directory. C ⬅ MEMORY p.3 PGDIR	7-6

G

Name, Key, or Label	Description, Type, and Keys	Page
ph	Phot, illuminance (795.774715459 cd/m^2) U ⬅ UNITS p.3 LIGHT PH	D-6
PICK	Copies object in level *n* to level 1. C PRG STK PICK	3-19
PICK	Copies object in current level to level 1. O ... ↑STK PICK	3-11
PICT	Returns *PICT* to level 1. C PRG DSPL PICT	19-30
PIXOFF	Turns off specified pixel in *PICT*. C PRG DSPL p.2 PIXOF	19-25
PIXON	Turns on specified pixel in *PICT*. C PRG DSPL p.2 PIXON	19-25
PIX?	Tests whether specified pixel in *PICT* is on or off. C PRG DSPL p.2 PIX?	19-26
pk	Peck, volume (.0088097675 m^3). U ⬅ UNITS VOL p.4 PK	D-6
PKT	Sends KERMIT commands to a server. C ⬅ I/O p.2 PKT	33-18
⬅ PLOT ➡ PLOT	Selects PLOT menu. O ⬅ PLOT Selects PLOT PLOTR menu. O ➡ PLOT	D-5 D-5
PLOT	Makes the selected entry the current statistical matrix and displays the third page of the STAT menu. O ⬅ STAT CAT PLOT	21-6
PLOTR	Selects PLOT PLOTR menu. ⬅ PLOT PLOTR ⬅ PLOT CAT PLOTR ➡ ALGEBRA PLOTR O ⬅ SOLVE CAT PLOTR	D-5

G

Name, Key, or Label	Description, Type, and Keys	Page
PMAX	Sets upper-right plot coordinates. C Must be typed in.	
PMIN	Sets lower-left plot coordinates. C Must be typed in.	
POLAR	Switches between rectangular and polar coordinates. O ⮡ POLAR	11-1
POLAR	Selects POLAR plot type. C ... PTYPE POLAR	19-13
POS	Returns the position of substring in string or object in list. C PRG OBJ p.3 POS	4-16
POWR	Selects UNITS POWR (power) menu. O ⮢ UNITS p.2 POWR	D-6
PREDV	Predicted value (same as PREDY). C Must be typed in.	
PREDX	Returns predicted value for independent variable, given value of dependent variable. C ⮢ STAT p.4 PREDX	21-12
PREDY	Returns predicted value for dependent variable, given value of independent variable. C ⮢ STAT p.4 PREDY	21-12
PRESS	Selects UNITS PRESS (pressure) menu. O ⮢ UNITS p.2 PRES	D-6
⮢ PREV ⮡ PREV	Selects previous page of menu. O ⮢ PREV Selects first page of menu. O ⮡ PREV	2-12 2-12
PRG	Selects PRG (program) menu. O PRG	D-3

Name, Key, or Label	Description, Type, and Keys	Page
(PRINT)	Selects PRINT menu. O (⬅)(PRINT)	D-4
PRLCD	Prints display. C (⬅)(PRINT) PRLCD O Simultaneously press (ON) (MTH)	32-4
PROB	Selects MTH PROB (probability) menu. O (MTH) PROB	D-3
PROMPT	Displays prompt string in status area and halts program execution. C (PRG) CTRL p.2 PROM	29-2
PRST	Prints all objects on stack. C (⬅)(PRINT) PRST	32-4
PRSTC	Prints all objects on stack in compact format. C (⬅)(PRINT) PRSTC	32-4
PRVAR	Prints name and contents of one or more variables (including port names). C (⬅)(PRINT) PRVAR	32-4
PR1	Prints object in level 1. (⬅)(PRINT) PR1 C (➡)(PRINT)	32-4
psi	Pounds per square inch, pressure (6894.75729317 kg/m·s^2). U (⬅)(UNITS) p.2 PRESS PSI	D-6
pt	Pint, volume (.000473176473 m^3). U (⬅)(UNITS) VOL p.2 PT	D-6
PTYPE	Selects PLOT PTYPE menu. (⬅)(PLOT) PTYPE ... PLOTR p.2 PTYPE O (➡)(PLOT) p.2 PTYPE	D-5
PURGE	Purges one or more specified variables. C (⬅)(PURGE)	6-8

Name, Key, or Label	Description, Type, and Keys	Page
PURGE	Purges one or more specified variables. If only one untagged variable specified, saves previous contents for recovery by LASTARG. O ◄ PURGE	6-9
PURG	Purges selected equation. ◄ SOLVE CAT p.2 PURG ► PLOT CAT p.2 PURG O ► ALGEBRA CAT p.2 PURG Purges selected statistical matrix. O ◄ STAT CAT p.2 PURG Purges selected alarm. ◄ TIME CAT PURG O ► TIME PURG	17-8 21-7 24-12
PUT	Replaces element in array or list. C PRG OBJ p.4 PUT	4-16
PUTI	Replaces element in array or list and increments index. C PRG OBJ p.4 PUTI	4-16
PVARS	Returns list of current backup objects and libraries within a port. C ◄ MEMORY p.2 PVARS	34-17
PVIEW	Displays *PICT* with specified pixel at upper-left corner of display. C PRG DSPL PVIEW	19-28
PWRFIT	Set curve-fitting model to Power. C ◄ STAT p.4 MODL PWR	21-11
PX→C	Converts pixel coordinates to user-unit coordinates. C PRG DSPL p.2 PX→C	19-9
→Q	Converts number to fractional equivalent. C ◄ →Q	9-5

G

Name, Key, or Label	Description, Type, and Keys	Page
QUAD	Finds solutions of first or second order polynomial. C ⬅ ALGEBRA QUAD	22-3
QUOTE	Returns argument expression unevaluated. F ⬅ ALGEBRA p.2 QUOT	8-2
qt	Quart, volume (.000946352946 m^3). U ⬅ UNITS VOL p.2 QT	D-6
→Qπ	Calculates and compares quotients of number and number/π. C ⬅ ALGEBRA p.2 →Qπ	9-5
r	Radian, plane angle (.1591549343092). U ⬅ UNITS p.3 ANGL R	D-6
R	Roentgen, radiation exposure (.000258 A·s/kg). U ⬅ UNITS p.3 RAD p.2 R	D-6
°R	Degrees Rankine, temperature. U ⬅ UNITS p.2 TEMP °R	D-6
rad	Rad, absorbed dose (.01 m^2/s^2). U ⬅ UNITS p.3 RAD RAD	D-6
RAD	Sets Radians mode. C ⬅ MODES p.3 RAD	9-8
RAD	Switches Radians and Degrees mode. O ⬅ RAD	9-8
RAD	Selects UNITS RAD (radiation) menu. O ⬅ UNITS p.3 RAD	D-6
RAND	Returns random number. C MTH PROB RAND	9-13
RATIO	Prefix form of / used internally by EquationWriter application. F Must be typed in.	

G

Name, Key, or Label	Description, Type, and Keys	Page
RCEQ	Returns equation in *EQ* to level 1. ⬅ PLOT ➡ STEQ ⬅ SOLVE ➡ STEQ ... PLOTR ➡ DRAW C ➡ PLOT ➡ DRAW	17-11
RCL	Recalls object stored in specified variable to stack. C ➡ RCL	6-5
RCL	Inserts algebraic from level 1 into EquationWriter equation. O ➡ RCL	16-21
RCLALARM	Recalls specified alarm from system alarm list. C ⬅ TIME ALRM p.2 RCLAL	24-16
RCLF	Returns binary integer representing states of system flags. C ➡ MODES p.2 RCLF	28-4
RCLKEYS	Returns list of current user-key assignments. C ➡ MODES RCLK	15-10
RCLMENU	Returns menu number of current menu. C ➡ MODES p.2 RCLM	29-19
RCLΣ	Recalls current statistical matrix in *ΣDAT*. C ⬅ STAT ➡ STOΣ	21-5
RCWS	Recalls binary integer wordsize. C MTH BASE RCWS	14-2
rd	Rod, length (5.0292100584 m). U ⬅ UNITS LENG p.3 RD	D-6
RDM	Redimensions array. C MTH MATR RDM	20-17
RDZ	Sets random number seed. C MTH PROB RDZ	9-13

G

Name, Key, or Label	Description, Type, and Keys	Page
RE	Returns real part of complex number or array. F (MTH) PARTS RE	11-11
RECN	Waits for stack-specified data from remote source running Kermit software. C (⬑)(I/O) p.2 RECN	33-18
RECV	Waits for sender-specified data from remote source running Kermit software. C (⬑)(I/O) RECV	33-17
rem	Rem, dose equivalent (.01 m^2/s^2). U (⬑)(UNITS) p.3 RAD REM	D-6
REPEAT	Begins loop clause. C (PRG) BRCH p.2 REPEA	27-12
REPL	Replaces portion of object with another like object. (PRG) OBJ p.3 REPL C (PRG) DSPL p.3 REPL	 4-17 19-29
REPL	Replaces portion of *PICT* with level-1 graphics object. ... DRAW p.3 REPL ... AUTO p.3 REPL O (⬑)(GRAPH) p.3 REPL	19-27
REPL	Replaces specified subexpression with algebraic from stack. O (⬑)(EQUATION) (◀) REPL	16-22
RES	Sets spacing between plotted points. ... PLOTR p.2 RES C (⬏)(PLOT) p.2 RES	19-4
(⬏) RES	Recalls spacing to stack. ... PLOTR p.2 (⬏) RES O (⬏)(PLOT) p.2 (⬏) RES	19-5

G

Name, Key, or Label	Description, Type, and Keys	Page
RESET	Resets plot parameters in *PPAR* in the current directory to their default states and erases and resizes *PICT*. ... PLOTR p.2 RESET O (→)(PLOT) p.2 RESET	18-16
RESTORE	Replaces *HOME* directory with backup copy. C (←)(MEMORY) p.3 RESTO	34-19
(REVIEW)	Displays statistical data in *ΣDAT*. O (←)(STAT) (←)(REVIEW) Displays current equation and plot parameters. ... PLOTR (←)(REVIEW) (→)(PLOT) (←)(REVIEW) ... DRAW (←)(REVIEW) ... AUTO (←)(REVIEW) O (←)(GRAPH) (←)(REVIEW) Displays current equation. O (←)(SOLVE) (←)(REVIEW) (←)(PLOT) (←)(REVIEW) Displays current equation and values of SOLVR variables. O ... SOLVR (←)(REVIEW) Displays unit names corresponding to selected menu. O (←)(UNITS) ... (←)(REVIEW) Displays pending alarm. O (←)(TIME) (←)(REVIEW) In other menus: Lists operation names and types. O (←)(REVIEW)	 21-5 18-16 18-21 17-11 18-7 17-13 13-6 24-5 6-7
RL	Rotates left by one bit. C (MTH) BASE p.2 RL	14-6

G

Name, Key, or Label	Description, Type, and Keys	Page
RLB	Rotates left by one byte. C (MTH) BASE p.2 RLB	14-6
RND	Rounds fractional part of number or name. F (MTH) PARTS p.4 RND	9-15
RNRM	Calculates row norm of array. C (MTH) MATR p.2 RNRM	20-17
ROLL	"Rolls up" n levels of the stack, level n+1 to level 1. C (PRG) STK ROLL	3-19
ROLL	"Rolls up" stack, pointer level to level 1. O ... ↑STK ROLL	3-11
ROLLD	"Rolls down" n levels of the stack, level 2 to level n. C (PRG) STK ROLLD	3-19
ROLLD	"Rolls down" stack, level 1 to pointer level. O ... ↑STK ROLLD	3-11
ROOT	Solves for unknown variable in equation. C (←)(SOLVE) ROOT	17-11
ROOT	Moves graphics cursor to intersection of function plot and x-axis, displays value of root, returns value to stack. O (←)(GRAPH) FCN ROOT	18-26
ROT	Moves object in level 3 to level 1. C (PRG) STK ROT	3-19
+ROW	Inserts row of zeros at current row. O (→)(MATRIX) p.2 +ROW	20-7
-ROW	Deletes current row. O (→)(MATRIX) p.2 -ROW	20-7
RPT	Selects TIME ALRM RPT (alarm repeat) menu. O (←)(TIME) ALRM RPT	D-5

G

Name, Key, or Label	Description, Type, and Keys	Page
RR	Rotates right by one bit. C (MTH) BASE p.2 RR	14-6
RRB	Rotates right by one byte. C (MTH) BASE p.2 RRB	14-6
RSD	Calculates correction to solution of system of equations. C (MTH) MATR RSD	20-18
RULES	Activates RULES transformation menu for specified object. O (←)(EQUATION) (◄) RULES	22-12
R→B	Real-to-binary conversion. C (MTH) BASE p.2 R→B	14-5
R→C	Real-to-complex conversion. C (PRG) OBJ pg.2 R→C	4-17
R→D	Radians-to-degrees conversion. F (MTH) VECTR p.2 R→D	9-11
R∡Z	Selects Polar/Cylindrical mode. (MTH) VECTR R∡Z O (←)(MODES) p.3 R∡Z	12-3
R∡∡	Selects Polar/Spherical mode. (MTH) VECTR R∡∡ O (←)(MODES) p.3 R∡∡	12-3
s	Second, time (1 s). U (←)(UNITS) TIME S	D-6
S	Siemens, electric conductance ($1\ A^2{\cdot}s^3/kg{\cdot}m^2$). U (←)(UNITS) p.2 ELEC p.2 S	D-6
SAME	Tests two objects for equality. C (PRG) TEST SAME	26-2
sb	Stilb, luminance ($10000\ cd/m^2$) U (←)(UNITS) p.3 LIGHT SB	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
SBRK	Sends serial break. C ⬑ I/O p.3 SBRK	33-20
SCALE	Sets scale of PLOT axes. ... PLOTR p.2 SCALE C ➦ PLOT p.2 SCALE	18-10
➦ SCALE	Recalls scale to stack. ... PLOTR p.2 ➦ SCALE O ➦ PLOT p.2 ➦ SCALE	18-16
SCATRPLOT	Draws scatter plot of statistical data in ΣDAT. C ⬑ STAT p.3 SCATR	21-19
SCATTER	Selects SCATTER plot type. C ... PTYPE p.2 SCATT	19-13
SCI	Selects Scientific display mode. C ⬑ MODES SCI	2-15
SCLΣ	Autoscales data in ΣDAT for scatter plot. C Must be typed in.	
SCONJ	Conjugates contents of variable. C ➦ MEMORY p.2 SCON	6-10
SDEV	Calculates standard deviation. C ⬑ STAT p.2 SDEV	21-9
SEC	Sets alarm repeat interval to n seconds. O ⬑ TIME ALRM RPT SEC	24-5
SEC+	Increments current time by 1 second. O ⬑ TIME ADJST SEC+	24-4
SEC-	Decrements current time by 1 second. O ⬑ TIME ADJST SEC-	24-4
SEND	Sends contents of variable to another device. C ⬑ I/O SEND	33-17

Name, Key, or Label	Description, Type, and Keys	Page
SERVER	Puts HP 48 into Kermit Server mode. [◀] [I/O] SERV C [▶] [I/O]	33-18
SET	Selects TIME SET menu. O [◀] [TIME] SET	D-5
SET	Sets alarm. O [◀] [TIME] ALRM SET	24-5
SETUP	Selects I/O SETUP menu. O [◀] [I/O] SETUP	D-4
SF	Sets specified flag. [PRG] TEST p.3 SF C [▶] [MODES] p.2 SF	28-2
SHOW	Reconstructs expression to resolve implicit variable name. C [◀] [ALGEBRA] SHOW	22-7
SIGN	Returns sign of number. F [MTH] PARTS SIGN	9-15
SIN	Sine. A [SIN]	9-9
SINH	Hyperbolic sine. A [MTH] HYP SINH	9-6
SINV	Replaces contents of variable with its inverse. C [▶] [MEMORY] p.2 SINV	6-10
SIZE	Finds dimensions of list, array, string, algebraic object, or graphics object. [PRG] OBJ p.3 SIZE C [PRG] DSPL p.2 SIZE	 4-17 19-28
←SKIP	Moves cursor left to next logical break. [◀] [EDIT] ←SKIP O ... EDIT ←SKIP	3-8

G

Name, Key, or Label	Description, Type, and Keys	Page
SKIP→	Moves cursor right to next logical break. ⬅ EDIT SKIP→ O ... EDIT SKIP→	3-8
SL	Shifts left by one bit. C MTH BASE p.3 SL	14-6
SLB	Shifts left by one byte. C MTH BASE p.3 SLB	14-6
SLOPE	Calculates and displays slope of function at cursor position, returns slope to stack. O ... FCN SLOPE	18-26
slug	Slug, mass (14.5939029372 kg). U ⬅ UNITS MASS SLUG	D-6
SNEG	Negates contents of variable. C ➡ MEMORY p.2 SNEG	6-10
⬅ SOLVE ➡ SOLVE	Selects SOLVE menu. O ⬅ SOLVE Selects SOLVR menu. O ➡ SOLVE	17-11 17-17
SOLVR	Selects SOLVR menu. ⬅ SOLVE SOLVR ⬅ SOLVE CAT SOLVR ➡ SOLVE ⬅ PLOT CAT SOLVR O ➡ ALGEBRA SOLVR	17-17
SPC	Types a blank space in command line. O SPC	3-3
SPEED	Selects UNITS SPEED menu. O ⬅ UNITS SPEED	D-6
SQ	Returns square of level-1 object. A ⬅ x^2	9-3

G

Name, Key, or Label	Description, Type, and Keys	Page
SR	Shifts right by one bit. C (MTH) BASE p.3 SR	14-6
sr	Steradian, solid angle ($7.95774715459 \times 10^{-2}$). U (←)(UNITS) p.3 ANGL SR	D-6
SRB	Shifts right by one byte. C (MTH) BASE p.3 SRB	14-6
SRECV	Reads specified number of characters from I/O port. C (←)(I/O) p.3 SRECV	33-20
SST	Single-steps through suspended program. O (PRG) CTRL SST	25-24
SST↓	Single-steps through suspended program and its subroutines. O (PRG) CTRL SST↓	25-24
st	Stere, volume (1 m^3). U (←)(UNITS) VOL ST	D-6
St	Stokes, kinematic viscosity (.0001 m^2/s) U (←)(UNITS) p.3 VISC ST	D-6
START (←) START (→) START	Begins definite loop. C (PRG) BRCH START Types START NEXT. O (PRG) BRCH (←) START Types START STEP. O (PRG) BRCH (→) START	27-2 27-3 27-5
(←)(STAT) (→)(STAT)	Selects STAT (statistics) menu. O (←)(STAT) Selects page 2 of STAT menu. O (→)(STAT)	D-5 D-5
STD	Selects Standard display mode. C (←)(MODES) STD	2-15

G

Name, Key, or Label	Description, Type, and Keys	Page
STEP	Ends definite loop. C [PRG] BRCH p.2 STEP	27-4 27-8
STEQ	Stores level 1 equation in *EQ*. [←][PLOT] STEQ ... PLOTR [←] DRAW [→][PLOT] [←] DRAW C [←][SOLVE] STEQ	17-5
STIME	Sets serial transmit/receive timeout. C [←][I/O] p.3 STIME	33-20
STK	Selects PRG STK (program stack) menu. O [PRG] STK	D-3
STK	Switches Last Stack recovery on and off. O [←][MODES] p.2 STK	15-11
↑STK	Selects Interactive Stack. ... EDIT ↑STK [←][EDIT] ↑STK [→][VISIT] ↑STK O [→][MATRIX] p.2 ↑STK	 3-10 20-7
→STK	Copies selected equation to level 1. ... CAT p.2 →STK O [→][ALGEBRA] p.2 →STK Copies selected matrix to level 1. O [←][STAT] CAT →STK Copies selected alarm to level 1. [←][TIME] CAT →STK O [→][TIME] →STK Copies selected matrix element to level 1. O [→][MATRIX] p.2 →STK	17-8 21-7 24-14 20-7
STO	Stores object in variable. C [STO]	6-2

Name, Key, or Label	Description, Type, and Keys	Page
STO	Stores object in variable and saves previous contents of variable for recovery by LASTARG. O STO Returns EquationWriter equation or *PICT* to stack. O STO	6-9 16-3 18-21
STOALARM	Stores level 1 alarm in system alarm list. C ⬅ TIME ALRM p.2 STOAL	24-16
STOF	Sets state of system and user flags. C ➡ MODES p.2 STOF	28-4
STOKEYS	Makes multiple user-key assignments. C ➡ MODES STOK	15-6
STO+	Adds contents of specified variable and specified number or other object. C ➡ MEMORY STO+	6-10
STO−	Calculates difference between contents of specified variable and specified number or other object. C ➡ MEMORY STO-	6-10
STO*	Multiplies contents of specified variable and specified number or other object. C ➡ MEMORY STO*	6-10
STO/	Calculates quotient of contents of specified variable and specified number or other object. C ➡ MEMORY STO/	6-10
STOΣ	Stores current statistics matrix in *ΣDAT*. C ⬅ STAT STOΣ	21-5
STR→	Converts string to component objects. C Must be typed in.	

G

Name, Key, or Label	Description, Type, and Keys	Page
→STR	Converts object into string. C (PRG) OBJ →STR	4-17
STWS	Sets binary integer wordsize. C (MTH) BASE STWS	14-1
SUB	Extracts specified portion of list or string, or graphics object. (PRG) OBJ p.3 SUB C (PRG) DSPL p.3 SUB	 4-18 19-29
SUB	Returns specified portion of *PICT* to stack. ... DRAW p.3 SUB ... AUTO p.3 SUB O (←)(GRAPH) p.3 SUB	19-27
SUB	Returns specified subexpression to stack. O (←)(EQUATION) (◄) SUB	22-12
Sv	Sievert, dose equivalent (.01 m^2/s^2) U (←)(UNITS) p.3 RAD SV	D-6
SWAP	Exchanges objects in levels 1 and 2. C (←)(SWAP)	3-4
SYM	Switches Symbolic and Numerical Results mode. O (←)(MODES) SYM	9-16
SYSEVAL	Evaluates system object. *Use only as specified by HP applications.* C Must be typed in.	
t	Metric ton, mass (1000 kg). U (←)(UNITS) MASS p.2 T	D-6
T	Tesla, magnetic flux (1 kg/As^2). U (←)(UNITS) p.2 ELEC p.2 T	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
←T	Move term left. O ⬅ EQUATION ◀ RULES ←T	22-14
➡ ←T	Executes ←T until no change in subexpression. O ⬅ EQUATION ◀ RULES ➡ ←T	22-18
T→	Move term right. O ⬅ EQUATION ◀ RULES T→	22-14
➡ T→	Executes T→ until no change in subexpression. O ⬅ EQUATION ◀ RULES ➡ T→	22-18
%T	Returns percent fraction that level 1 is of level 2. F MTH PARTS p.2 %T	9-7
→TAG	Combines objects in levels 1 and 2 to create tagged object. C PRG OBJ →TAG	4-18
TAN	Tangent. A TAN	9-9
TANH	Hyperbolic tangent. A MTH HYP TANH	9-6
TAYLR	Calculates Taylor's polynomial. C ⬅ ALGEBRA TAYLR	23-8
tbsp	Tablespoon, volume ($1.47867647813 \times 10^{-5}$ m^3). U ⬅ UNITS VOL p.3 TBSP	D-6
TEMP	Selects UNITS TEMP (temperature) menu. O ⬅ UNITS p.2 TEMP	D-6
TEST	Selects PRG TEST (program test) menu. O PRG TEST	D-3
TEXT	Displays stack display. C PRG DSPL p.4 TEXT	19-29

G

Name, Key, or Label	Description, Type, and Keys	Page
THEN	Begins true clause. C (PRG) BRCH p.2 THEN	26-5
therm	EEC therm, energy (105506000 kg·m^2/s^2) U (←)(UNITS) p.2 ENRG p.2 THER	D-6
TICKS	Returns system time as binary integer in units of clock ticks. C (←)(TIME) p.2 TICKS	24-19
TIME	Returns current time as a number. C (←)(TIME) p.2 TIME	24-19
(←)(TIME) (→)(TIME)	Selects TIME menu. O (←)(TIME) Selects Alarm Catalog. O (→)(TIME)	D-5 24-12
TIME	Selects UNITS TIME menu. O (←)(UNITS) TIME	D-6
→TIME	Sets system time. C (←)(TIME) SET →TIM	24-2
>TIME	Sets alarm time. O (←)(TIME) ALRM >TIME	24-5
TLINE	Switches pixels on line defined by coordinates in levels 1 and 2. C (PRG) DSPL TLINE	19-25
TLINE	Switches pixels on and off on line between mark and cursor. ... DRAW p.2 TLINE ... AUTO p.2 TLINE O (←)(GRAPH) p.2 TLINE	19-23
TMENU	Displays list-defined menu but does not change contents of *CST*. C (→)(MODES) p.2 TMEN	29-18
ton	Short ton, mass (907.18474 kg). U (←)(UNITS) MASS p.2 TON	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
tonUK	Long (UK) ton, mass (1016.0469088 kg). U ⬑ UNITS MASS p.2 TONU	D-6
torr	Torr (mmHg), pressure (133.322368421 kg/ms^2). U ⬑ UNITS p.2 PRESS TORR	D-6
TOT	Sums each column of matrix in ΣDAT. C ⬑ STAT p.2 TOT	21-9
TRANSIO	Selects one of three character translation settings. C ⬑ I/O SETUP TRAN	33-4
TRG*	Expands trigonometric and hyperbolic functions of sums and differences. O ⬑ EQUATION ◀ RULES TRG*	22-17
→TRG	Replace exponential with trigonometric functions. O ⬑ EQUATION ◀ RULES →TRG	22-17
TRN	Transposes matrix. C MTH MATR TRN	20-17
TRNC	Truncates (rounds down) number in level 2 as specified in level 1. F MTH PARTS p.4 TRNC	9-15
TRUTH	Selects TRUTH plot type. C ... PTYPE TRUTH	19-13
tsp	Teaspoon, volume (4.92892159375 × 10^{-6} m^3). U ⬑ UNITS VOL p.3 TSP	D-6
TSTR	Converts date and time in number form to string form. C ⬑ TIME p.2 TSTR	24-17
TVARS	Returns variables containing specified object type. C ⬑ MEMORY p.2 TVARS	4-19

G

Name, Key, or Label	Description, Type, and Keys	Page
TYPE	Returns type-number of argument object. PRG OBJ p.2 TYPE C PRG TEST TYPE	4-19
u	Unified atomic mass (1.66057×10^{-27} kg). U ← UNITS MASS p.3 u	D-6
UBASE	Converts unit object to SI base units. F → UNITS UBASE	13-11
UFACT	Factors specified compound unit. C → UNITS UFACT	13-13
→UNIT	Combines objects in levels 1 and 2 to create unit object. PRG OBJ p.2 →UNIT C → UNITS →UNIT	4-18
← UNITS → UNITS	Selects UNITS Catalog menu. O ← UNITS Selects UNITS Command menu. O → UNITS	D-6 D-6
UNTIL	Begins test clause. C PRG BRCH p.2 UNTIL	27-10
UPDIR	Makes parent directory the current directory. C ← UP	7-5
USR	Turns User mode on and off. O ← USR	15-5
UTPC	Returns probability that chi-square random variable is greater than x. C MTH PROB p.2 UTPC	21-21
UTPF	Returns probability that Snedecor's F random variable is greater than x. C MTH PROB p.2 UTPF	21-21

Name, Key, or Label	Description, Type, and Keys	Page
UTPN	Returns probability that normal random variable is greater than x. C (MTH) PROB p.2 UTPN	21-21
UTPT	Returns probability that Student's t random variable is greater than x. C (MTH) PROB p.2 UTPT	21-21
UVAL	Returns scalar of specified unit object. F (→)(UNITS) UVAL	13-22
V	Volt, electrical potential (1 kg·m^2/A·s^3). U (←)(UNITS) p.2 ELEC V	D-6
VAR	Calculates variance of statistical data columns in ΣDAT. C Must be typed in.	21-9
(VAR)	Selects VAR (variables) menu. O (VAR)	6-3
1-VAR	Makes the selected entry the current statistical matrix and displays the second page of the STAT menu. O (←)(STAT) CAT 1-VAR	21-6
2-VAR	Makes the selected entry the current statistical matrix and displays the fourth page of the STAT menu. O (←)(STAT) CAT 2-VAR	21-6
VARS	Returns list of variables in current directory. C (←)(MEMORY) VARS	6-8
VEC	Switches vector and array modes. O (→)(MATRIX) VEC	20-7
VECTR	Selects MTH VECTR (math vector) menu. O (MTH) VECTR	D-3

G

Name, Key, or Label	Description, Type, and Keys	Page
VIEW	Copies object in current level into appropriate environment for viewing. (▲) VIEW O ... ↑STK VIEW	3-11
	Displays selected equation. O ... CAT VIEW	17-8
	Displays selected matrix. O (⬅)(STAT) CAT VIEW	21-7
	Displays selected alarm. O (⬅)(TIME) CAT VIEW	24-12
(➡) VIEW	Copies object stored in variable in the current level into appropriate environment for viewing. O ... ↑STK (➡) VIEW	3-11
VISC	Selects UNITS VISC (viscosity) menu. O (⬅)(UNITS) p.3 VISC	D-6
(VISIT)	If argument is name, copies contents of associated variable into command line for editing. If argument is a stack level number, copies object in that level into command line for editing. O (➡)(VISIT)	3-7
VOL	Selects UNITS VOL (volume) menu. O (⬅)(UNITS) VOL	D-6
VTYPE	Returns type number of object stored in local or global name. C (PRG) OBJ p.2 VTYPE	4-19
→V2	Combines two real numbers into a 2-D vector or complex number. C (MTH) VECTR p.2 →V2	12-14

Name, Key, or Label	Description, Type, and Keys	Page
→V3	Combines three real numbers into 3-D vector. C (MTH) VECTR p.2 →V3	12-14
V→	Separates 2- or 3-element vector according to current angle mode. C (MTH) VECTR p.2 V→	12-14
W	Watt, power (1 kg·m^2/s^3) (←)(UNITS) p.2 POWR W U (←)(UNITS) p.2 ELEC W	D-6
*W	Adjusts horizontal plot scale. C ... PLOTR p.3 *W	19-6
WAIT	Halts program execution for specified number of seconds or until key pressed. C (PRG) CTRL p.2 WAIT	29-16
Wb	Weber, magnetic flux (1 kg·m^2/A·s^2). U (←)(UNITS) p.2 ELEC p.2 WB	D-6
WEEK	Sets alarm repeat interval to n weeks. O (←)(TIME) ALRM RPT WEEK	24-5
WHILE	Begins indefinite loop. C (PRG) BRCH WHILE	27-12
(←)WHILE	Types WHILE REPEAT END O (PRG) BRCH (←)WHILE	27-13
WID→	Increases column width and decrements number of columns. O (→)(MATRIX) WID→	20-7
←WID	Decreases column width and increments number of columns. O (→)(MATRIX) ←WID	20-7
X	Selects x-axis zoom. O ... ZOOM X	18-22

G

Name, Key, or Label	Description, Type, and Keys	Page
ΣX	Returns sum of data in independent column in *ΣDAT*. C (←)(STAT) p.5 ΣX	21-20
ΣX^2	Returns sum of squares of data in independent column in *ΣDAT*. C (←)(STAT) p.5 ΣX^2	21-20
XAUTO	Selects x-axis zoom with autoscaling. O ... ZOOM XAUTO	18-22
XCOL (→) XCOL	Specifies independent-variable column in matrix in *ΣDAT*. C (←)(STAT) p.3 XCOL Recalls independent-variable column number to stack. O (←)(STAT) p.3 (→) XCOL	21-11 21-11
XMIT	Without Kermit protocol, performs serial send of string. C (←)(I/O) p.3 XMIT	33-20
XOR	Logical or binary exclusive OR. (MTH) BASE p.4 XOR F (PRG) TEST XOR	 14-6 26-3
XPON	Returns exponent of number. F (MTH) PARTS p.3 XPON	9-15
XRNG (→) XRNG	Specifies x-axis display range. ... PLOTR XRNG C (→)(PLOT) XRNG Recalls x-axis display range to stack. ... PLOTR (→) XRNG O (→)(PLOT) (→) XRNG	18-9 18-15
XROOT	Returns level 1 root of the real number in level 2. A (→)($\sqrt[x]{y}$)	9-3
XY	Selects x- and y-axis zoom. O ... ZOOM XY	18-22

Name, Key, or Label	Description, Type, and Keys	Page
XYZ	Selects Rectangular mode. (MTH) VECTR XYZ O (←)(MODES) p.3 XYZ	12-3
ΣX*Y	Returns sum of products of data in independent and dependent columns in *ΣDAT*. C (←)(STAT) p.5 ΣX*Y	21-20
Y	Selects *y*-axis zoom. O ... ZOOM Y	18-22
ΣY	Returns sum of data in dependent column in *ΣDAT*. C (←)(STAT) p.5 ΣY	21-20
ΣY^2	Returns sum of squares of data in dependent column in *ΣDAT*. C (←)(STAT) p.5 ΣY^2	21-20
YCOL	Selects indicated column of *ΣDAT* as dependent-variable column for two-variable statistics. C (←)(STAT) p.3 YCOL	21-11
(→) YCOL	Recalls dependent-variable column number to stack. O (←)(STAT) p.3 (→) YCOL	21-11
yd	International yard, length (.9144 m). U (←)(UNITS) LENG YD	D-6
yd^2	Square yard, area (.83612736 m^2). U (←)(UNITS) AREA YD^2	D-6
yd^3	Cubic yard, volume (.764554857984 m^3). U (←)(UNITS) VOL YD^3	D-6
yr	Year, time (31556925.9747 s). U (←)(UNITS) TIME YR	D-6

G

Name, Key, or Label	Description, Type, and Keys	Page
YRNG	Specifies y-axis display range. ... PLOTR YRNG C (→)(PLOT) YRNG	18-9
(→) YRNG	Recalls y-axis display range to stack. ... PLOTR (→) YRNG O (→)(PLOT) (→) YRNG	18-16
Z-BOX	Zooms in to box whose opposite corners are defined by mark and cursor. ... DRAW Z-BOX ... AUTO Z-BOX O (←)(GRAPH) Z-BOX	18-34
(←) Z-BOX	Zooms to box, autoscaling y-axis. ... DRAW (←) Z-BOX ... AUTO (←) Z-BOX O (←)(GRAPH) (←) Z-BOX	18-34
ZOOM	Selects GRAPHICS ZOOM menu. ... DRAW ZOOM ... AUTO ZOOM O (←)(GRAPH) ZOOM	18-22
+	Adds two objects. A (+)	4-12 9-3
(+/-)	If cursor is on a number, changes sign of mantissa or exponent of that number. Otherwise, acts as NEG key. C (+/-)	2-7
+/-	Switches cursor style between super-imposing and inverting cross. ... DRAW p.3 +/- ... AUTO p.3 +/- O (←)(GRAPH) p.3 +/-	18-20
+1-1	Add and subtract 1. O (←)(EQUATION) (◀) RULES +1-1	22-14

G

Name, Key, or Label	Description, Type, and Keys	Page
−	Subtracts two objects. A [−]	9-3
−( )	Double negate and distribute. O [←][EQUATION] [◀] RULES −( )	22-16
*	Multiplies two objects. A [×]	9-3
*1	Multiply by 1. O [←][EQUATION] [◀] RULES *1	22-13
/	Divides two objects. A [÷]	9-3
/1	Divide by 1. O [←][EQUATION] [◀] RULES /1	22-13
^	Raises number to specified power. A [y^x]	9-3
^1	Raise to power 1. O [←][EQUATION] [◀] RULES ^1	22-13
<	"Less-than" comparison. [PRG] TEST p.2 < F [α][←][2]	26-2
≤	"Less-than-or-equal" comparison. [PRG] TEST p.2 ≤ F [α][←][3]	26-2
>	"Greater-than" comparison. [PRG] TEST p.2 > F [α][→][2]	26-2
≥	"Greater-than-or-equal" comparison. [PRG] TEST p.2 ≥ F [α][→][3]	26-2
=	"Equals" function. A [←][=]	8-6

Name, Key, or Label	Description, Type, and Keys	Page
==	"Equality" comparison. PRG TEST p.2 == F α ← 1	26-2
≠	"Not-equal" comparison. PRG TEST p.2 ≠ F α ← 0	26-2
α	Turns alpha-entry mode on and off. O α	2-8
{}	Switches implicit parentheses on and off. O ← EQUATION ← {}	16-11
" "	Returns equation to stack as string. O ← EQUATION → " "	16-4
°	Degree, plane angle ($2.77777777778 \times 10^{-3}$). U ← UNITS p.3 ANGL °	D-6
!	Factorial. MTH PROB ! F α ← DEL	9-13
∫	Integral. A → ∫	23-11
∂	Derivative. A → ∂	23-1
Ω	Ohm, electric resistance ($1\ kg{\cdot}m^2/A^2{\cdot}s^3$). U ← UNITS p.2 ELEC Ω	D-6
%	Returns level 2 percent of level 1. A MTH PARTS p.2 %	9-7
π	Symbolic constant π (3.14159265359). F ← π	9-15
Σ	Summation. F → Σ	23-5

G

Name, Key, or Label	Description, Type, and Keys	Page
Σ+	Adds data point to matrix in *ΣDAT*. C [←] [STAT] Σ+	21-2
Σ−	Subtracts data point from matrix in *ΣDAT*. C [←] [STAT] [←] Σ+	21-4
√	Returns square root of level-1 object. A [√x]	9-3
|	Appends local name, or variable of integration, and its value to evaluated expression. F [←] [ALGEBRA] p.2 |	22-26
1/( )	Double-invert and distribute. O [←] [EQUATION] [◄] RULES 1/( )	22-16
12/24	Switches between 12-hour and 24-hour display formats. O [←] [TIME] SET 12/24	24-2
(( ))	Parenthesize neighbors. O [←] [EQUATION] [◄] RULES (( ))	22-15
(←	Expand-subexpression-left. O [←] [EQUATION] [◄] RULES (←	22-15
[→] (←	Executes (← until no change in subexpression. O [←] [EQUATION] [◄] RULES [→] (←	22-18
→( )	Distribute prefix function. O [←] [EQUATION] [◄] RULES →( )	22-15
→)	Expand-subexpression-right. O [←] [EQUATION] [◄] RULES →)	22-15
[→] →)	Executes →) until no change in subexpression. O [←] [EQUATION] [◄] RULES [→] →)	22-18
←→	Commute arguments. O [←] [EQUATION] [◄] RULES ←→	22-15

G

Name, Key, or Label	Description, Type, and Keys	Page
→	Creates local variables. C [right-shift] [→]	25-13
[left-shift]	Left shift key. O [left-shift]	2-5
[right-shift]	Right shift key. O [right-shift]	2-5
[◂]	With no command line, drops object in level 1.	3-5
	In command line, deletes character to left of cursor. O [◂]	2-7
	Deletes contents of current stack level. O ... ↑STK [◂]	3-12
[▲]	With no or one-line command line: Activates Interactive Stack.	3-10
	In multi-line command line: Moves cursor up one line.	3-16
	In Interactive Stack: Moves pointer up one level.	3-12
	In Graphics environment: Moves cursor up one pixel.	18-20
	In scrolling mode: Moves window up one pixel.	16-3
	In MatrixWriter application: Moves cell cursor up one row.	20-6
	In EquationWriter application: Starts numerator.	16-3
	In Selection environment: Moves cursor up one object.	22-11
	In catalogs: Moves pointer up one entry. O [▲]	

Name, Key, or Label	Description, Type, and Keys	Page
⬅ ▲	In catalogs: Moves pointer up one page.	
	In Interactive Stack: Moves pointer up 4 levels.	3-12
	O ⬅ ▲	
➡ ▲	In multi-line command line: Moves cursor to top line.	3-16
	In Interactive Stack: Moves pointer to highest numbered stack level.	3-12
	In Graphics environment: Moves cursor to top edge of *PICT*.	18-20
	In MatrixWriter application: Moves cell cursor to top element of current column.	20-6
	In Selection environment: Moves cursor to topmost object.	22-12
	In catalogs: Moves pointer to top of list.	
	O ➡ ▲	
▼	With no or one-line command line: Activates "best" editor.	3-7
	In multi-line command line: Moves cursor down one line.	3-16
	In Interactive Stack: Moves pointer down one level.	3-12
	In Graphics environment: Moves cursor down one pixel.	18-20
	In scrolling mode: Moves window down one pixel.	16-3
	In MatrixWriter application: Moves cell cursor down one row.	20-6
	In EquationWriter application: Ends subexpression.	16-3
	In Selection environment: Moves cursor down one object.	22-11
	In catalogs: Moves pointer down one entry.	
	O ▼	

Name, Key, or Label	Description, Type, and Keys	Page
◀ ▼ (left-shift, down arrow)	In catalogs: Moves pointer down page. In Interactive Stack: Moves pointer down 4 levels. O ◀ ▼	 3-12
▶ ▼ (right-shift, down arrow)	In multi-line command line: Moves cursor to bottom line. In Interactive Stack: Moves pointer to level 1. In Graphics environment: Moves cursor to bottom edge of *PICT*. In MatrixWriter application: Moves cell cursor to last element of current column. In EquationWriter application: Ends all subexpressions. In Selection environment: Moves cursor to bottommost object. In catalogs: Moves pointer to end of list. O ▶ ▼	3-16 3-12 18-20 20-6 16-3 22-12
◀ (left arrow)	With no command line: Enter Graphics environment. In command line: Moves cursor one character left. In Graphics environment: Moves cursor one pixel left. In scrolling mode: Moves window left one pixel. In MatrixWriter application: Moves cell cursor one column left. In EquationWriter application: Activates Selection environment. In Selection environment: Moves cursor one object left. O ◀	18-19 3-16 18-20 16-3 20-6 22-11 22-11

G

Name, Key, or Label	Description, Type, and Keys	Page
⬑ ◀ (⬑ GRAPH)	In EquationWriter application and Graphics environments: Invokes scrolling mode. O ⬑ ◀ (⬑ GRAPH)	16-3 18-20
⮣ ◀	In command line: Moves cursor to start of current line. In Graphics environment: Moves cursor to left edge of *PICT*. In MatrixWriter application: Moves cell cursor to first element of current row. In Selection environment: Moves cursor to leftmost object. O ⮣ ◀	3-16 18-20 20-6 22-12
▶	With no command line: Exchanges the objects in levels 1 and 2. In command line: Moves cursor one character right. In Graphics environment: Moves cursor one pixel right. In scrolling mode: Moves window right one pixel. In MatrixWriter application: Moves cell cursor one column right. In EquationWriter application: Ends subexpression. In Selection environment: Moves cursor one object right. O ▶	3-4 3-16 18-20 16-3 20-6 16-3 22-11

Name, Key, or Label	Description, Type, and Keys	Page
ⓡ ▶	In command line: Moves cursor to end of current line.	3-16
	In Graphics environment: Moves cursor to right edge of *PICT*.	18-20
	In MatrixWriter application: Moves cell cursor to last element of current row.	20-6
	In EquationWriter application: Ends all subexpressions.	16-3
	In Selection environment: Moves cursor to rightmost object.	22-12
	O ⓡ ▶	

G

Index

Special characters

A

C

E

F

G

S

T

V

Contacting Hewlett-Packard

For Information about Using the Calculator. If you have questions about how to use the calculator, first check the table of contents, the subject index, and "Answers to Common Questions" in appendix A. If you can't find an answer in the manual, you can contact the Calculator Support Department:

Hewlett-Packard
Calculator Support
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.
(503) 757-2004
8:00 a.m. to 3:00 p.m. Pacific time
Monday through Friday

For Service. If your calculator doesn't seem to work properly, see appendix A for diagnostic instructions and information on obtaining service. If you are in the United States and your calculator requires service, mail it to the Corvallis Service Center:

Hewlett-Packard
Corvallis Service Center
1030 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.
(503) 757-2002

If you are outside the United States, see appendix A for information on locating the nearest service center.

HP Calculator Bulletin Board System. The Bulletin Board provides for the exchange of software and information among HP calculator users, developers, and distributors. It operates at 300/1200/2400 baud, full duplex, no parity, 8 bits, 1 stop bit. The telephone number is (503) 750-4448. The Bulletin Board is a free service—you pay for only the long-distance telephone charge.

HEWLETT PACKARD

Part Number 00048-90091
Edition 1
English
Printed in U.S.A. 11/91

For HP internal use only:
00048-90092