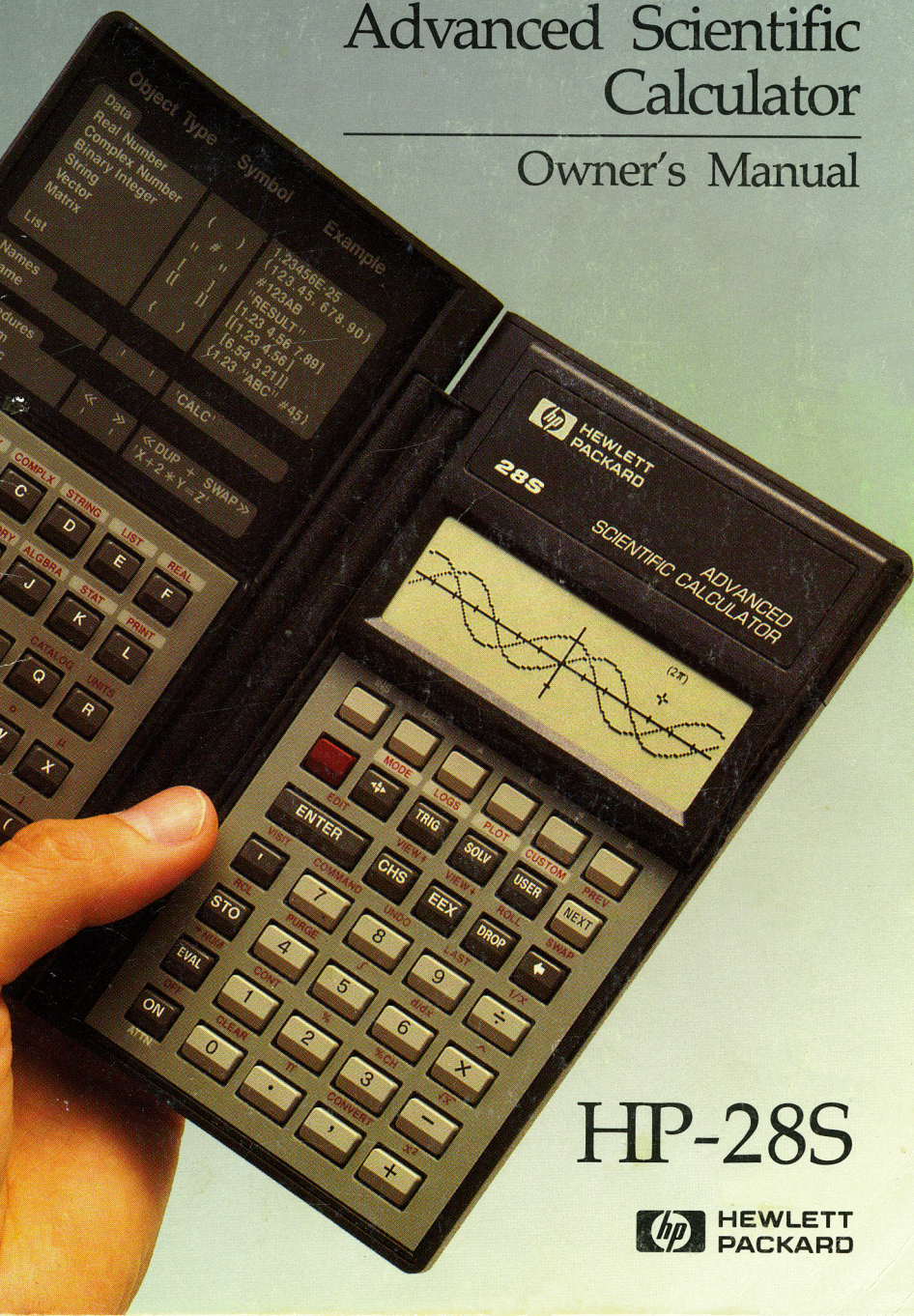# HEWLETT-PACKARD

Advanced Scientific Calculator

Owner's Manual

HP-28S

HEWLETT PACKARD

HP-28S
Advanced Scientific Calculator

Owner's Manual

HEWLETT PACKARD

Notice

For warranty and regulatory information for this calculator, see pages 291 and 295.

Printing History

Welcome to the HP-28S

Congratulations! With the HP-28S you can easily solve complicated problems, including problems you couldn't solve on a calculator before. The HP-28S combines powerful numerical computation with a new dimension—*symbolic computation.* You can formulate a problem symbolically, find a symbolic solution that shows the global behavior of the problem, and obtain numerical results from the symbolic solution.

The HP-28S offers the following features:

- Algebraic manipulation. You can expand, collect, or rearrange terms in an expression, and you can symbolically solve an equation for a variable.

- Calculus. You can calculate derivatives, indefinite integrals, and definite integrals.

- Numerical solutions. Using HP Solve on the HP-28S, you can solve an expression or equation for any variable. You can also solve a system of linear equations. With multiple data types, you can use complex numbers, vectors, and matrices as easily as real numbers.

- Plotting. You can plot expressions, equations, and statistical data.

- Unit conversion. You can convert between any equivalent combinations of the 120 built-in units. You can also define your own units.

- Statistics. You can calculate single-sample statistics, paired-sample statistics, and probabilities.

- Binary number bases. You can calculate with binary, octal, and hexadecimal numbers and perform bit manipulations.

- Direct entry for algebraic formulas, plus RPN logic for interactive calculations.

The *HP-28S Owner's Manual* (this manual) contains three parts. Part 1, "Fundamentals," demonstrates how to work some simple problems. Part 2, "Summary of Calculator Features," builds on part 1 to help you apply those examples to your own problems. Part 3, "Programming," describes programming features and demonstrates them in a series of programming examples.

The *HP-28S Reference Manual* gives detailed information about commands. It is a dictionary of menus, describing the concepts and commands for each menu.

We recommend that you first work through the examples in part 1 of the Owner's Manual to get comfortable with the calculator, and then look at part 2 to gain a broader understanding of the calculator's operation. When you want to know more about a particular command, look it up in the Reference Manual. When you want to learn about programming, read part 3 of the Owner's Manual.

These manuals show you how to use the HP-28S to do math, but they don't teach math. We assume that you're already familiar with the relevant mathematical principles. For example, to use the calculus features of the HP-28S effectively, you should know elementary calculus.

On the other hand, you don't need to understand all the math topics in the HP-28S to use those parts of interest to you. For example, you don't need to understand calculus to use the statistical capabilities.

Contents

Part 2: Summary of Calculator Features

Part 3: Programming

Appendixes & Indexes

How To Use This Manual

If you have the time and inclination, you can read this manual from front to back, working every example. If not, we recommend the following approach for getting started.

1. Read the first five chapters in part 1, "Fundamentals," to get comfortable with the calculator.

2. There are two appendixes that compare the HP-28S with other styles of calculators.

   ■ If you're familiar with other Hewlett-Packard calculators that use RPN, read appendix B, "Notes for RPN Calculator Users," on page 296.

   ■ If you're familiar with calculators that use a form of algebraic entry, read appendix C, "Notes for Algebraic Calculator Users," on page 302.

3. If you're interested in a topic covered later in part 1, you can skip ahead and try the examples in that chapter.

What's in This Manual

Part 1, "Fundamentals," demonstrates how to work some simple problems. While solving these problems you'll learn the basics about HP-28S operations, object types, and menus.

Part 2, "Summary of Calculator Features," builds on part 1. It provides more detail about how to use the calculator, including options and features not discussed in part 1. Using part 2, you can extend the examples in part 1 to solve your own problems.

Part 3, "Programming," describes the programming features of the HP-28S. The last chapter, "Programming Examples," contains a series of short programs that demonstrate programming techniques.

For More Information

As you work the examples in this manual, you may have questions about the features demonstrated or mentioned in the examples. Both this manual and the Reference Manual contain additional information.

- If you have problems, see "Answers to Common Questions" on page 282.

- For a brief description of what each key does, see "Key Index" on page 327.

- For a brief description of the commands in each menu, see appendix D, "Menu Map," on page 306.

- For detailed information about a menu, look in the Reference Manual. All menus (plus some additional topics) appear in alphabetical order. The contents of the dictionary are listed on the back cover of the Reference Manual.

- For detailed information about a particular command, look in the "Operation Index" at the back of the Reference Manual. There you'll find a reference to a dictionary entry (usually a menu) and a page reference to the particular command.

Part 1

Fundamentals

1

Getting Started

This chapter first describes the calculator's basic features, then demonstrates a simple calculation. Next, an annotated illustration of the keyboard highlights the major features of the keyboard and display. Last, you'll learn about the catalog of commands, which is a handy guide to commands and how to use them.

Preliminaries

This section describes the calculator's basic features.

Opening and Closing the Case

The calculator forms its own case, opening and closing like a book. To open the calculator, hold it with the hinge away from you and open it with your thumbs.
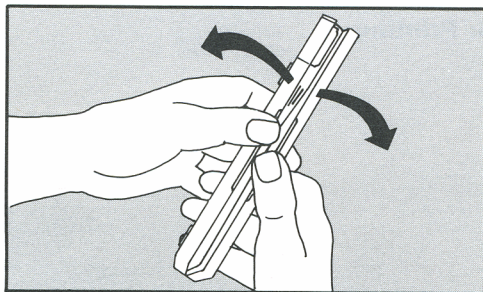
To close the calculator, fold the two sides together and press until you hear a click.

You can fold back the left-hand side of the calculator until it is back-to-back with the right-hand keyboard. This is handy for field work—when you want to hold the calculator in one hand and operate it with the other—or to save space on a desk.
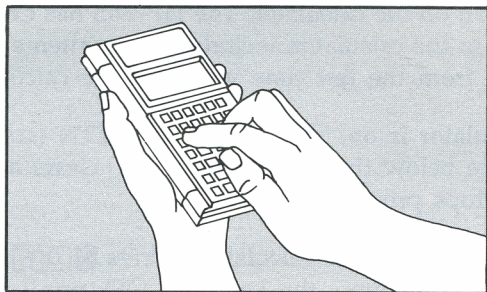
Locating the Battery Door and Printer Port

With the calculator open, note the location of the battery door and the printer port.
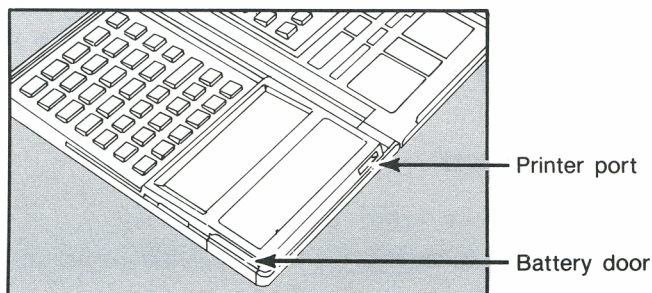
The HP-28S is powered by three N-cell alkaline batteries. Batteries are included with the calculator. If the batteries are not already installed, follow the instructions that start on page 286.

When you use the HP-28S with a printer, the calculator sends information to the printer via an infra-red signal. This signal is emitted from the printer port and received by the printer. Printer operations are described in chapter 15.

Turning the HP-28S On and Off

Press [ON] to turn on the calculator. The HP-28S has *Continuous Memory*, so all data in the calculator, including the contents of the display, are unchanged from the last time you used the calculator.

While the calculator is on, [ON] acts as the ATTN (*attention*) key, as printed in white below the key. Pressing [ON] clears any text you've typed in and stops programs.

Press ■[OFF] to turn off the calculator. ("Press ■[OFF]" means "press the shift key ■, then press the key with OFF printed above it.")

If the calculator is inactive for about 10 minutes, it automatically turns off to conserve energy. Press [ON] to turn it on again.

Clearing All Memory (Memory Reset)

You can restore the calculator to its initial state by resetting memory. All information in the calculator is lost. Any modes you've changed (number display format, angle mode, and so on) are restored to their *default* settings.

To reset memory:

1. Press and hold [ON].
2. Press and hold [INS] (in the upper-left corner of the right-hand keyboard).
3. Press and release [▶] (in the upper-right corner of the right-hand keyboard).
4. Release [INS].
5. Release [ON].

The calculator beeps and displays Memory Lost. The message automatically disappears when you press a key.

If you begin to reset memory but change your mind, *continue holding down* [ON] while you press [DEL] (in the top row, next to [INS]), and then release [ON]. Pressing [DEL] cancels the reset sequence.

Adjusting the Display Contrast

You can adjust the contrast of the display to compensate for various viewing angles and light intensities.

To adjust the contrast:

1. Press and hold [ON].
2. Press [+] one or more times to darken the display, or press [−] one or more times to lighten the display.
3. Release [ON].

Keyboard Calculations

Try the following calculation.

$$(15 + 23) \times \sin 30°$$

The basic steps are the same as using paper and pencil. First you'll calculate 15 + 23, which produces an intermediate result. Next you'll calculate sin 30°, which produces the other intermediate result. Finally, you'll combine the intermediate results for the answer.

If you make a mistake while keying in a number, you can:

■ Press [◀] to erase the last digit you keyed in.
■ Press [ON] to erase all the digits you keyed in.

Start with a clean sheet of paper.

█ CLEAR

```
4:
3:
2:
1:
```

The display shows the *stack*, which is your work area. Currently the stack is empty.

Press ⒈ ⒌ to write 15 in the *command line*.

15

```
3:
2:
1:
15□
```

Note that the stack moves up to make room for the command line, so only three stack levels are displayed.

Put 15 on the stack.

ENTER

```
4:
3:
2:
1:          15
```

The number goes in *stack level 1*, as indicated by 1: at the left. Note that the command line disappears, so four stack levels are displayed again.

Press ⒉ ⒊ to write 23 in the command line.

23

```
3:
2:
1:          15
23□
```

Put 23 in level 1.

ENTER

```
4:
3:
2:          15
1:          23
```

The number 15, which was in level 1, is lifted to level 2.

Add 15 and 23.

[+]

```
4:
3:
2:
1:              38
```

The numbers 15 and 23 are removed from the stack, and their sum, 38, is returned to level 1. You'll leave this intermediate result on the stack while you calculate the second intermediate result.

To calculate sin 30° you'll use the TRIG (*trigonometry*) menu.

[TRIG]

```
3:
2:
1:              38
SIN | ASIN | COS | ACOS | TAN | ATAN
```

The bottom line of the display shows six commands in the TRIG menu. The six *menu labels* (SIN through ATAN) define the six *menu keys* (the keys immediately below the display).

Press [3][0] to write 30 in the command line.

30

```
2:
1:              38
30□
SIN | ASIN | COS | ACOS | TAN | ATAN
```

Put 30 in level 1.

[ENTER]

```
3:
2:              38
1:              30
SIN | ASIN | COS | ACOS | TAN | ATAN
```

The previous result, 15 + 23 = 38, is lifted to level 2.

Calculate sin 30°.

SIN

```
3:
2:              38
1:              .5
SIN | ASIN | COS | ACOS | TAN | ATAN
```

The number 30 is removed from level 1, and its sine, .5, is returned to level 1. The previous result, 38, remains in level 2.

Calculate 38 × .5.

⌧

```
3:
2:
1:                                    19
SIN  ASIN  COS  ACOS  TAN  ATAN
```

The numbers 38 and .5 are removed from levels 1 and 2, and their product, 19, is returned to level 1.

This completes the calculation:

$$(15 + 23) \times \sin 30° = 19.$$

To summarize, here's a general procedure for the calculation you just completed.

1. Key a number into the *command line.*

2. Press ENTER to put a number on the *stack.*

3. Press the key to execute the command. (If the command doesn't appear on the keyboard, select the *menu* that contains the command, and press the *menu key* below the appropriate *menu label*.)

The previous example demonstrated that all calculations occur on the stack. To highlight this idea, you pressed ENTER to put every number on the stack. In practice, *you need to press* ENTER *only to separate two numbers keyed in sequentially*—in the example, to separate 15 and 23. Try repeating the example, omitting the second and third ENTER s.

The style of calculation illustrated above, in which you enter numbers onto the stack before you perform mathematical functions, is called RPN (*Reverse Polish Notation*), postfix notation, or stack logic. Nearly all HP-28S commands, not just calculations, use stack logic. This system uses two simple rules:

- The inputs required by a function, called the *arguments* to the function, must be on the stack before the function is executed.

- The results of a function are returned to the stack, where they are available as arguments to the next function.

You can also calculate by entering an expression in algebraic form, as it might appear written in a book. In the next chapter you'll perform the same calculation as above, using an algebraic expression.

An Overview of the Calculator

This section points out some major features of the calculator, including a catalog of commands that lists and describes each command.

Major Features and Concepts

The illustrations on pages 26 and 27 show the calculator keyboard and display, with important features identified. The numbers in the following descriptions correspond to the numbers in the illustrations.

Object Type	Symbol	Example
Data		
Real Number		1.23456E-25
Complex Number	( )	(123.45,678.90)
Binary Integer	#	#123AB
String	" "	"RESULT"
Vector	[ ]	[1.23 4.56 7.89]
Matrix	[[ ]]	[[1.23 4.56] [6.54 3.21]]
List	{ }	{1.23 "ABC" #45}
Names		
Name	` `	'CALC'
Procedures		
Program	« »	«DUP + SWAP»
Algebraic	` `	'X + Z · Y = Z'
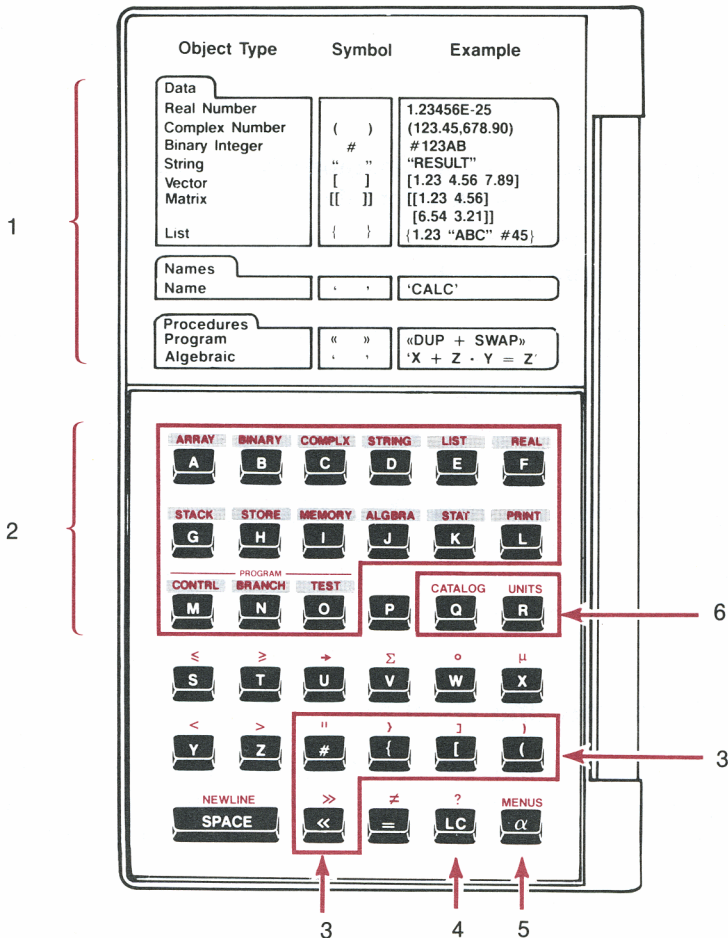
1. Object types and formats
2. Menu selection (shifted)
3. Object delimiters
4. Lowercase
5. Entry mode
6. Command and unit listings (shifted)

7. Annunciators
8. Command line
9. Shift key
10. Enter command line
11. Delimiter for symbolic objects
12. Power on and off; clear command line; stop program
13. Number entry
14. Arithmetic
15. Backspace
16. Menu selection, next menu row
17. Menu keys
18. Menu labels
19. Stack levels

1. Object types and formats. This table shows the correct delimiters and examples for the 10 basic types of object. An "object" is any of the individual items you work with on the calculator. The 10 basic object types are:

- Real numbers, such as 5 or -4.3×10^{15}.

- Complex numbers, which are a pair of real numbers representing a complex number $x + iy$ or a point in a plane.

- Binary integers, which are unsigned integers used in computer science.

- Strings, which contain arbitrary sequences of characters.

- Vectors, which are one-dimensional arrays used in linear algebra.

- Matrices, which are two-dimensional arrays used in linear algebra.

- Lists, which contain arbitrary sequences of objects.

- Names, which enable you to name and store other objects and to perform symbolic calculations.

- Programs, which enable you to create your own commands.

- Algebraics, which represent mathematical expressions and equations.

2. Menu selection (shifted). Use the menu selection keys to assign commands to the menu keys. For example, press ■ ARRAY to select the ARRAY menu. To select a different menu, press another menu selection key.

There are additional menu selection keys on the right-hand keyboard (see item 16).

3. Object delimiters. These symbols identify the different object types (see item 1). For example, # identifies binary integers, while « and ■ » identify programs.

Real numbers require no delimiters. Symbolic objects (names and algebraics) require the ' delimiter, located on the right-hand keyboard (see item 11).

4. Lowercase. Press LC to key in lowercase letters. Lowercase mode continues until you press LC a second time, press ENTER to process the command line, or press ON to clear the command line.

5. Entry mode. The command line has three entry modes, each suited to entering certain types of objects. Entry modes change automatically as you key in objects, but sometimes you want manual control; the $\boxed{\alpha}$ key enables you to select the entry mode you want.

6. Command and unit listings (shifted). Press ■ $\boxed{\text{CATALOG}}$ for a listing of all HP-28S commands and their required arguments (page 31). Press ■ $\boxed{\text{UNITS}}$ for a listing of the units recognized in unit conversion (page 141).

7. Annunciators. The annunciators indicate the status of the calculator. When an annunciator is visible, it indicates the following:

Annunciator	Meaning
O	Suspended program.
↴	Shift key ■ was pressed.
α	Alpha entry mode.
((•))	Busy, not ready for input.
▭	Low battery.
(2π)	Radians mode.
▱	Sending data to printer.

8. Command line. The text you key in goes in the command line.

9. Shift key. Press the colored shift key ■ to execute a command printed in color above a key.

10. Enter command line. Press $\boxed{\text{ENTER}}$ to process the text in the command line.

11. Delimiter for symbolic objects. Delimiters are punctuation that identify types of objects; symbolic objects are names and algebraics. To key in a symbolic object, press $\boxed{'}$ at the beginning and (when necessary) the end of the object.

Real numbers require no delimiters. The delimiters for other object types are on the left-hand keyboard (see items 1 and 3).

12. Power on and off; clear command line; stop program. To turn on the calculator, press [ON]; to turn it off, press ■[OFF]. (OFF is printed on the keyboard above [ON]. "Press ■[OFF]" means press the shift key ■ and then press [ON].)

While the calculator is on, [ON] also acts as the ATTN (*attention*) key to clear text in the command line or stop a running program. (ATTN is printed on the keyboard below [ON].)

13. Number entry. To key in numbers, use the digit keys [0] through [9], [CHS] (*change sign*), and [EEX] (*enter exponent*). Assuming you want to use the period as the decimal point (page 36), use [·] to separate the integer part from the fractional part. Number entry is described on page 39.

14. Arithmetic. The arithmetic functions are described in "One-Number Functions" on page 40 and "Two-Number Functions" on page 41.

15. Backspace. Press [◀] to erase the last character you typed in.

16. Menu selection, next menu row. Use the menu selection keys to assign commands to the menu keys. For example, press [TRIG] to select the TRIG menu. To select a different menu, press another menu selection key.

When no menu labels are visible, the *cursor menu* is active. The operations in the cursor menu ([INS] through [▶]) are printed in white above the menu keys. When menu labels are visible, press [◀▶] to select the cursor menu. To restore the previous menu, press [◀▶] a second time.

A menu can contain more than one row, with up to six commands in each row. Press [NEXT] to display the next row of the current menu. Press ■[PREV] to display the previous row.

There are additional menu selection keys on the left-hand keyboard (see item 2). For an alphabetical listing of all menus, including a brief description of the commands in each menu, refer to appendix D, "Menu Map."

17. Menu keys. The menu keys are defined by the menu labels. If no labels are visible, these keys execute the cursor menu operations labeled in white above the keys.

18. Menu labels. The menu labels show the current definitions of the menu keys.

19. Stack levels. The stack shows the objects you're currently working with. Each numbered stack level (level 1, level 2, and so on) holds one object.

The Catalog of Commands

The HP-28S contains a catalog of all commands, listed alphabetically. For each command the catalog shows its usage—that is, the arguments required by the command. For a complete description of any command listed in the catalog, refer to "Operation Index" in the back of the Reference Manual.

Start the catalog.

■ CATALOG

```
┌─────────────────────────────────────┐
│             ABORT                    │
│                                      │
│                                      │
│ NEXT PREV      USE FETCH QUIT        │
└─────────────────────────────────────┘
```
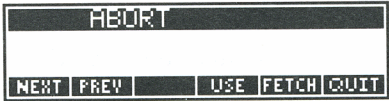
The first command is ABORT.

Normal calculator operation is suspended while the catalog is active. The NEXT and PREV operations move the catalog to other commands. The USE operation displays the arguments required by the current command. The FETCH and QUIT operations terminate the catalog, restoring normal calculator operation.

Try pressing NEXT and PREV to move through the catalog. You can hold down the keys for repeated moves.

You can move to the first catalog entry for a particular letter by pressing the letter key. Try "T".

[T]

```
┌─────────────────────────────────────┐
│               TAN                    │
│                                      │
│                                      │
│ NEXT │ PREV │      │ USE │FETCH│QUIT │
└─────────────────────────────────────┘
```

The first "T" command is the TAN function. If you press a symbol (non-letter) key on the left-hand keyboard, the catalog moves to the first catalog entry for that symbol. Try "Σ".

■ [Σ]

```
┌─────────────────────────────────────┐
│               Σ+                     │
│                                      │
│                                      │
│ NEXT │ PREV │      │ USE │FETCH│QUIT │
└─────────────────────────────────────┘
```

The first "Σ" command is the Σ+ command. If you press a symbol key on the left-hand keyboard, and no commands begin with that symbol, the catalog moves to +, the first non-alphabetical command.

[#]

```
┌─────────────────────────────────────┐
│                +                     │
│                                      │
│                                      │
│ NEXT │ PREV │      │ USE │FETCH│QUIT │
└─────────────────────────────────────┘
```

Check the usage for +.

USE

```
┌─────────────────────────────────────┐
│ USAGE: +                             │
│ 2: Real Number                       │
│ 1: Real Number                       │
│ NEXT │ PREV │      │      │     │QUIT │
└─────────────────────────────────────┘
```

This shows that you can add two real numbers. Check the next combination.

NEXT

```
┌─────────────────────────────────────┐
│ USAGE: +                             │
│ 2: Real Number                       │
│ 1: Complex Number                    │
│ NEXT │ PREV │      │      │     │QUIT │
└─────────────────────────────────────┘
```

This shows that you can add real and complex numbers. Check the next combination.

NEXT

```
┌─────────────────────────────────────┐
│ USAGE: +                             │
│ 2: Complex Number                    │
│ 1: Real Number                       │
│ NEXT │ PREV │      │      │     │QUIT │
└─────────────────────────────────────┘
```

This shows that the real and complex numbers can be in either order.

Check the 14 remaining combinations. The last combination looks like this.

```
┌─────────────────────────────────┐
│ USAGE: +                        │
│ 2: Any Object                   │
│ 1: List                         │
│ NEXT PREV              QUIT     │
└─────────────────────────────────┘
```

When you're done checking combinations, return to the main catalog.

QUIT

```
┌─────────────────────────────────┐
│               +                 │
│                                 │
│                                 │
│ NEXT PREV       USE FETCH QUIT  │
└─────────────────────────────────┘
```

You can now move to another catalog entry and check its combinations of arguments. When you're done with the catalog, return to normal calculator operation.

QUIT

```
┌─────────────────────────────────┐
│ 3:                              │
│ 2:                              │
│ 1:                           19 │
│ SIN ASIN COS ACOS TAN ATAN      │
└─────────────────────────────────┘
```

Alternatively, you can exit the catalog by pressing FETCH , which also writes the name of the current command in the command line.

2

Doing Arithmetic

There are two ways to do arithmetic on the HP-28S. You can do arithmetic using the stack, as you did in the previous chapter, or you can enter an *expression* representing the calculation. In the previous chapter you calculated:

$$(15 + 23) \times \sin 30°$$

Here's how to make the same calculation using an expression.

Clear the stack and select the TRIG menu.

■ CLEAR TRIG

```
3:
2:
1:
SIN │ASIN│ COS │ACOS│ TAN │ATAN
```

Start the expression.

'
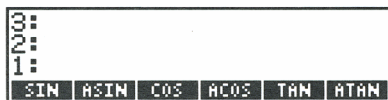
```
2:
1:
'▯
SIN │ASIN│ COS │ACOS│ TAN │ATAN
```

The cursor changes, indicating *Algebraic Entry mode*. You'll see the effects of this entry mode as you key in the expression.

Key in the first part of the expression.

(15 + 23 ■)

```
2:
1:
'(15+23)▯
SIN │ASIN│ COS │ACOS│ TAN │ATAN
```

Because of Algebraic Entry mode, pressing ⊕ wrote the character + in the command line rather than executing the command.

Continue the expression.

⊠ SIN

```
2:
1:
'(15+23)*SIN(目
SIN  ASIN  COS  ACOS  TAN  ATAN
```

Because of Algebraic Entry mode, pressing ⊠ wrote ✳ in the command line, and pressing SIN wrote SIN(in the command line, rather than executing the commands.

Complete the expression and put it on the stack.

30 ENTER

```
3:
2:
1:       '(15+23)*SIN(30)'
SIN  ASIN  COS  ACOS  TAN  ATAN
```

The closing parenthesis) and the closing delimiter ' are added for you.

Evaluate the expression.

EVAL

```
3:
2:
1:                         19
SIN  ASIN  COS  ACOS  TAN  ATAN
```

The expression is removed from the stack, and the result, 19, is returned to level 1.

This completes the calculation:

$$(15 + 23) \times \sin 30° = 19.$$

To perform a calculation that's already written as an expression, such as in a textbook, it's easier to key in the expression and evaluate it. Alternatively, to see the intermediate results of your calculation, or to perform an on-going calculation, it's easier to calculate on the stack. The results are the same.

The relationship between stack calculations and expressions is demonstrated in chapter 4, "Repeating Calculations." In that chapter you calculate on the stack, using names instead of numbers, to produce an expression.

Entering and Displaying Numbers

There are modes that affect how numbers are displayed. To demonstrate the choices, put the number ⅔ on the stack.

Put 2 on the stack.

2 [ENTER]

```
┌─────────────────────────┐
│3:                       │
│2:                    19 │
│1:                     2 │
│SIN│ASIN│COS│ACOS│TAN│ATAN│
└─────────────────────────┘
```

Divide by 3.

3 [÷]

```
┌─────────────────────────┐
│3:                       │
│2:                    19 │
│1:        .666666666667  │
│SIN│ASIN│COS│ACOS│TAN│ATAN│
└─────────────────────────┘
```

The result, ⅔, is returned to level 1. This result is the decimal approximation to ⅔, as displayed by the default choices for decimal point and number display mode. The next section describes other choices.

Changing the Decimal Point

In the United States a period is used to separate the integer part of a number from the fractional part. In this role the period is called a decimal point; the general term for this numerical punctuation is a *radix mark*.

Many other countries use a comma as the radix mark. You can select the comma as follows.

Select the MODE menu.

■[MODE]

```
┌─────────────────────────┐
│3:                       │
│2:                    19 │
│1:        .666666666667  │
│STD■│FIX│SCI│ENG│DEG■│RAD │
└─────────────────────────┘
```

The first row of the MODE menu appears. Display the second row of the MODE menu.

NEXT

```
3:
2:                              19
1:             .666666666667
CMD■ UNDO■ LAST■ ML■  RDX.  PRMD
```

Select the comma as the radix mark.

RDX,

```
3:
2:                              19
1:             ,666666666667
CMD■ UNDO■ LAST■ ML■  RDX.■ PRMD
```

The decimal points are replaced by comma radix marks, and a square appears in the menu label RDX, to indicate that RDX, mode is turned on.

Turn off RDX, mode to restore decimal points.

RDX,

```
3:
2:                              19
1:             .666666666667
CMD■ UNDO■ LAST■ ML■  RDX.  PRMD
```

Selecting Number Display Mode

You can choose how many decimal places are displayed.

Return to the first row of the MODE menu.

NEXT

```
3:
2:                              19
1:             .666666666667
STD■  FIX   SCI   ENG  DEG■  RAD
```

You can move from the last row in a menu to the first row by pressing NEXT. Since the MODE menu has only two rows, pressing NEXT cycled back to the first row.

The menu shows four basic choices for number display mode: STD (standard), FIX (fixed), SCI (scientific), and ENG (engineering). The label for STD currently includes a square, indicating that STD is the current choice. In STD mode the number of decimal places depends

on the value. For an integer, no decimal places are shown; for the example displayed above, the maximum of 12 decimal places are shown.

The other display formats show a given number of decimal places—from 0 through 11—regardless of the number being displayed. We'll demonstrate each of the other display formats with two decimal places. Only the displays of the numbers are rounded—internally, the numbers are unchanged.

Display ⅔ rounded to two decimal places.

2 FIX

```
3:
2:                    19.00
1:                     0.67
 STD │ FIX■│ SCI │ ENG │DEG■│ RAD
```

Display ⅔ as a *mantissa* and an *exponent*, with the mantissa rounded to two decimal places.

2 SCI

```
3:
2:                   1.90E1
1:                  6.67E-1
 STD │ FIX │SCI■│ ENG │DEG■│ RAD
```

The value of the number is the product of the mantissa and 10 raised to the power of the exponent. The mantissa is always between 1 and 9.99999999999.

Display ⅔ as a mantissa and an exponent, with the mantissa rounded to two decimal places and the exponent a multiple of 3.

2 ENG

```
3:
2:                   19.0E0
1:                  667.E-3
 STD │ FIX │ SCI │ENG■│DEG■│ RAD
```

Return to standard number display mode.

STD

```
3:
2:                       19
1:             .666666666667
 STD■│ FIX │ SCI │ ENG │DEG■│ RAD
```

Keying In Numbers

You can enter numbers as a mantissa and an exponent, where the value of the number is the product of the mantissa and 10 raised to the power of the exponent. The mantissa or the exponent or both can be negative.

For example, key in the number -4.2×10^{-12}.

First key in the digits for the mantissa.

4.2
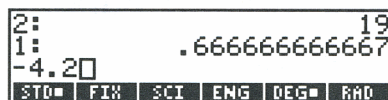
```
2:                        19
1:          .666666666667
4.2▯
 STO■  FIX   SCI   ENG  DEG■  RAD
```

If you make a mistake, press ◀ to erase the mistake and then key in the correct digits.

Next make the mantissa negative.
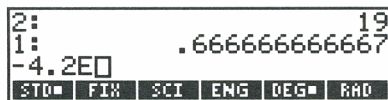
[CHS]

```
2:                        19
1:          .666666666667
-4.2▯
 STO■  FIX   SCI   ENG  DEG■  RAD
```

"CHS" stands for "change sign"—pressing [CHS] a second time would make the mantissa positive again.

Now begin the exponent.
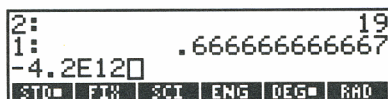
[EEX]

```
2:                        19
1:          .666666666667
-4.2E▯
 STO■  FIX   SCI   ENG  DEG■  RAD
```

"EEX" stands for "enter exponent." The E in the command line marks the number's exponent. If you press [EEX] by mistake for a number without an exponent, you can erase the E by pressing ◀, just as you would erase an incorrect digit.

Key in the digits for the exponent.

12

```
2:                        19
1:          .666666666667
-4.2E12▯
 STO■  FIX   SCI   ENG  DEG■  RAD
```

Make the exponent negative.

[CHS]

```
2:                        19
1:          .666666666667
-4.2E-12▯
STO▪│ FIX │ SCI │ ENG │DEG▪│ RAD
```

Put the number on the stack.

[ENTER]

```
3:                        19
2:          .666666666667
1:                -4.2E-12
STO▪│ FIX │ SCI │ ENG │DEG▪│ RAD
```

Don't forget to use [CHS] to key in negative numbers. For example, if this manual shows the keystrokes −4 [×], you'll need to press [4] [CHS] [×].

One-Number Functions

Functions that act on a single number—for example, negating a number or taking a square root—are called one-number functions. All act on the number in level 1. There are four one-number functions on the keyboard:

■ Press [CHS] to negate the number.

■ Press ▪[1/x] to take the inverse (reciprocal) of the number.

■ Press ▪[√x] to take the square root of the number.

■ Press ▪[x²] to square the number.

If you're keying in a number, it's not necessary to press [ENTER] before executing the one-number function—pressing the function key automatically performs ENTER for you. For example, you can calculate $\frac{1}{8}$ as follows:

8 ▪[1/x]

```
3:          .666666666667
2:                -4.2E-12
1:                    .125
STO▪│ FIX │ SCI │ ENG │DEG▪│ RAD
```

Two-Number Functions

Functions that act on two numbers—such as addition—are called two-number functions. All act on the numbers in levels 1 and 2.

When you're keying in both arguments to the function, as when you divided 2 by 3 on page 36, you must press [ENTER] to separate the two arguments. When one or both arguments are already on the stack from previous calculations, you don't need to press [ENTER].

Addition and Subtraction

Calculate 36 + 17.

36 [ENTER]
17 [+]

```
3:              -4.2E-12
2:                  .125
1:                    53
STD■ | FIX | SCI | ENG |DEG■| RAD
```

The result is 53.

For addition the order of the numbers doesn't matter. However, the order is important for subtraction. Next calculate 91 − 27.

91 [ENTER]
27 [−]

```
3:                  .125
2:                    53
1:                    64
STD■ | FIX | SCI | ENG |DEG■| RAD
```

The result is 64.

Multiplication and Division

Calculate 13 × 6.

13 [ENTER]
6 [×]

```
3:                    53
2:                    64
1:                    78
STD■ | FIX | SCI | ENG |DEG■| RAD
```

The result is 78.

For multiplication the order of the numbers doesn't matter. However, the order is important for division. Next calculate 182/14.

182 [ENTER]
14 [÷]

```
3:                    64
2:                    78
1:                    13
STD■ | FIX | SCI | ENG |DEG■| RAD
```

The result is 13.

Powers and Roots

The order of the numbers is important for both powers and roots. Calculate 5^3.

5 [ENTER]
3 ■ [^]

```
3:                    78
2:                    13
1:                   125
STD■ | FIX | SCI | ENG |DEG■| RAD
```

The result is 125.
To calculate $\sqrt[4]{2401}$, first put 2401 on the stack.

2401 [ENTER]

```
3:                    13
2:                   125
1:                  2401
STD■ | FIX | SCI | ENG |DEG■| RAD
```

Now raise 2401 to the ¼ power.

4 ■ [1/x] ■ [^]

```
3:                    13
2:                   125
1:                     7
STD■ | FIX | SCI | ENG |DEG■| RAD
```

The result is 7.

Percentages

Calculate 40% of 85.

85 [ENTER]
40 ■ [%]

```
3:                        125
2:                          7
1:                         34
STD■ | FIX | SCI | ENG |DEG■| RAD
```

The result is 34.

For "percent" the order of the numbers doesn't matter. However, the order is important for "percent change." Calculate the percent change from 60 to 75.

60 [ENTER]
75 ■ [%CH]

```
3:                          7
2:                         34
1:                         25
STD■ | FIX | SCI | ENG |DEG■| RAD
```

The result is +25, meaning that 75 is 25% more than 60.

Swapping Levels 1 and 2

For all the functions where the order of the numbers is important—subtraction, division, powers, roots, and percent change—you can switch the order of the numbers by pressing ■ [SWAP]. For example, you currently have 25 on the stack; suppose you want to calculate 30 − 25.

Key in 30.

30

```
2:                         34
1:                         25
30□
STD■ | FIX | SCI | ENG |DEG■| RAD
```

Swap the order of 25 and 30.

■ [SWAP]

```
3:                         34
2:                         30
1:                         25
STD■ | FIX | SCI | ENG |DEG■| RAD
```

Note that pressing [SWAP] performed ENTER for you.

Subtract 25 from 30.

⊟

```
3:                    7
2:                   34
1:                    5
STO■ | FIX | SCI | ENG | DEG■ | RAD
```

The result is 5.

Clearing Objects From the Stack

As you worked these examples, you accumulated quite a few numbers on the stack. The stack grows without limit as you put more objects on the stack, and those objects remain until you use them in an operation or until you clear them.

You can clear objects one at a time or all at once.

Clear the number in level 1.

[DROP]

```
3:                  125
2:                    7
1:                   34
STO■ | FIX | SCI | ENG | DEG■ | RAD
```

Objects in higher levels move down one level each.

Clear all objects from the stack.

■[CLEAR]

```
3:
2:
1:
STO■ | FIX | SCI | ENG | DEG■ | RAD
```

It's a good idea to clear the stack whenever you start a problem. As you work on the problem you'll know that all objects on the stack are relevant to the current problem, not left over from the previous problem.

Chain Calculations

When you perform complicated calculations, the stack acts as temporary storage to hold intermediate results. This temporary storage acts automatically. For example, suppose you want to calculate the total resistance of the following circuit:
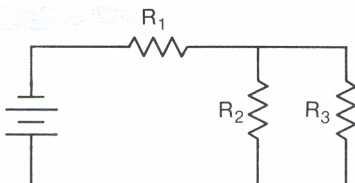
The formula for total resistance in this circuit is:

$$R_{total} = R_1 + \cfrac{1}{\cfrac{1}{R_2} + \cfrac{1}{R_3}}$$

If R_1, R_2, and R_3 have resistances of 8, 6, and 3 ohms respectively, calculate the following:

$$R_{total} = 8 + \cfrac{1}{\cfrac{1}{6} + \cfrac{1}{3}}$$

Calculate as follows:

Put 8 on the stack.

8 [ENTER]

You'll leave 8 on the stack until it's time to add it to the rest of the calculation.

Put ⅙ on the stack.

6 ■ 1/x

```
3:
2:                              8
1:              .16666666667
STD■  FIX   SCI  ENG  DEG■  RAD
```

Put ⅓ on the stack.

3 ■ 1/x

```
3:                              8
2:              .16666666667
1:              .333333333333
STD■  FIX   SCI  ENG  DEG■  RAD
```

Add the reciprocals of 6 and 3.

[+]

```
3:
2:                              8
1:                             .5
STD■  FIX   SCI  ENG  DEG■  RAD
```

Take the reciprocal of the sum.

■ 1/x

```
3:
2:                              8
1:                             2
STD■  FIX   SCI  ENG  DEG■  RAD
```

Complete the calculation of R$_{total}$.

[+]

```
3:
2:
1:                            10
STD■  FIX   SCI  ENG  DEG■  RAD
```

The result is 10 ohms.

If You Execute the Wrong Function

The HP-28S includes recovery features to help you "backtrack" when you mistakenly execute a function. The following steps reverse the effects of a function, whether a one-number or two-number function.

1. Press ▮ UNDO to recover the previous contents of the stack.
2. If a number was in the command line when you made the mistake, press ▮ COMMAND to recover the previous contents of the command line.
3. Continue the calculation.

3

Using Variables

Variables enable you to refer to objects by name. You create a variable by associating a name object with any other object. The name object defines the name of the variable; the other object defines the contents of the variable. You can then use the variable's name to refer to the variable's contents.

Variables are stored in *user memory*, a part of the calculator's memory distinct from the stack. The stack is designed for temporary storage of the objects you're currently working with, such as the numbers you're using in a calculation. User memory is designed for long-term storage of variables, such as numbers that you use repeatedly.

In this chapter you'll create a numerical variable, which may be a familiar concept to you; you'll also create a program variable, which is probably an unfamiliar concept. In the HP-28S, a program has no intrinsic name—it is simply another object type. You name the program by storing it in a variable, just as you would a number, and you can then execute the program by name.

The steps to create, recall, evaluate, change, rename, or purge a variable are identical for all variables, regardless of their contents. This uniformity makes the HP-28S both easy to use and powerful, because there are fewer special rules and because it is more flexible.

Introduction to Variables

The simplest variables are numerical variables. This section shows how to create, recall, and evaluate a numerical variable.

Creating a Numerical Variable

Suppose you repeatedly use a volume measurement of 133 in your calculations. Create a variable named VOL (for "volume") as follows:

Clear the stack and select the USER menu.

■ CLEAR   USER

```
3:
2:
1:
▮▮▮▮▮▮ ▮▮▮▮ ▮▮▮▮ ▮▮▮▮ ▮▮▮▮
```

The USER menu shows your variables. It's blank because you haven't created any variables yet.

Put the number in level 1.

133 ENTER

```
3:
2:
1:                    133
▮▮▮▮ ▮▮▮▮ ▮▮▮▮ ▮▮▮▮ ▮▮▮▮
```

Put the name 'VOL' in level 1.

' VOL ENTER

```
3:
2:                    133
1:                  'VOL'
▮▮▮▮ ▮▮▮▮ ▮▮▮▮ ▮▮▮▮ ▮▮▮▮
```

Note that the closing ' is added for you. The number 133 is lifted to level 2. (In practice you don't need to press ENTER, but it's included here for clarity.)

Create the variable VOL.

STO

```
3:
2:
1:
▮VOL▮ ▮▮▮▮ ▮▮▮▮ ▮▮▮▮ ▮▮▮▮
```

The number and the name are removed from the stack, creating a variable named VOL with a value of 133. Note that VOL now appears in the USER menu.

Recalling a Numerical Variable

Now that you've created the variable VOL, return its value to the stack.

Put the name VOL on the stack, taking advantage of the USER menu.

['] VOL [ENTER]

```
3:
2:
1:            'VOL'
VOL
```

Recall the contents of VOL.

■[RCL]

```
3:
2:
1:             133
VOL
```

This is the number you stored in VOL.

If you're accustomed to a calculator with storage registers, recalling is a familiar process. On the HP-28S, variables are recalled infrequently; more often they are *evaluated*.

Evaluating a Numerical Variable

For numerical variables, "evaluating" has the same meaning as "recalling"—evaluating a numerical variable returns the number to the stack. You'll see that evaluation is easier. (When you create a program variable later in this chapter, you'll see that evaluating and recalling can have quite different effects.)

Return the value of VOL to the stack by evaluation.

VOL

```
3:
2:             133
1:             133
VOL
```

You can also evaluate VOL by typing in its name *without quotes*.

VOL [ENTER]

```
3:             133
2:             133
1:             133
VOL
```

Changing the Value of a Variable

You can change the value of a variable by using the same procedure as when you created the variable. The new value replaces the old value.

Now change the value of the variable VOL to 151.

Write the new value in the command line.

151

```
2:                    133
1:                    133
151□
VOL
```

Note that the cursor appears as an empty box. The cursor will change in the next step.

Write the variable's name in the command line.

['] VOL

```
2:                    133
1:                    133
151'VOL█
VOL
```

The cursor changed when you pressed ['] to indicate the new *entry mode*—how the calculator responds when you press keys.

Initially the command line was in *immediate execution mode,* suitable for keyboard calculations. When you pressed ['], which indicates a name or an expression, the command line changed to *algebraic entry mode,* suitable for entering names and expressions:

■ Pressing a function key such as [+] writes the character + rather than executing the command.

■ Pressing a USER menu key writes the variable's name rather than evaluating the variable.

Now store the new value in the variable.

[STO]

```
3:                    133
2:                    133
1:                    133
VOL
```

Check the new value.

VOL

```
3:                    133
2:                    133
1:                    151
 VOL
```

Purging a Variable

When you finish with the variable VOL, purge it from user memory. Write the variable name in the command line.

['] VOL

```
2:                    133
1:                    151
'VOL▊
 VOL
```

(The quote ['] is necessary to avoid evaluating the variable.)

Purge the variable VOL from user memory.

■ [PURGE]

```
3:                    133
2:                    133
1:                    151
```

Note that the label for VOL disappears from the USER menu.

Changing the Name of a Variable

You can effectively change the name of a variable by creating a new variable with the same value and purging the original variable.

In this section you'll first go through the steps required to rename a variable, then write a program that contains the same steps, and finally store the program in a variable and execute it by name.

In preparation, create a variable so you have something to rename—a variable A with value 10.

Put the value 10 on the stack.

10 [ENTER]

```
3:                          133
2:                          151
1:                           10
```

Create the variable A.

['] A [STO]

```
3:                          133
2:                          133
1:                          151
    A
```

Note that A appears in the USER menu.

Suppose you want to rename A to B. Put the old name on the stack.

['] A [ENTER]

```
3:                          133
2:                          151
1:                          'A'
    A
```

Put the new name on the stack.

['] B [ENTER]

```
3:                          151
2:                          'A'
1:                          'B'
    A
```

This completes the preparation: the variable exists, the old name is on the stack, and the new name is on the stack. The old and new names are the *arguments* to the program—the program will assume they're on the stack in this order. The steps that follow are those that will be in the program.

The steps include three common stack-manipulation commands, OVER, ROT, and SWAP. You'll see how they work as you execute the steps.

Copy the old name to level 1. (Use the OVER command in the STACK menu.)

[STACK] OVER

```
3:                          'A'
2:                          'B'
1:                          'A'
 DUP  OVER  DUP2 DROP2  ROT  LIST→
```

Recall the contents of the variable.

■ [RCL]

```
3:                        'A'
2:                        'B'
1:                         10
 DUP | OVER | DUP2 |DROP2| ROT |LIST→
```

Move the old name to level 1. (Use the ROT command, for "rotate".)

ROT

```
3:                        'B'
2:                         10
1:                        'A'
 DUP | OVER | DUP2 |DROP2| ROT |LIST→
```

Purge the old variable. (By purging the old variable before creating the new one, you avoid making an extra copy of the value.)

■ [PURGE]

```
3:                        151
2:                        'B'
1:                         10
 DUP | OVER | DUP2 |DROP2| ROT |LIST→
```

Put the contents and the new name in the correct order.

■ [SWAP]

```
3:                        151
2:                         10
1:                        'B'
 DUP | OVER | DUP2 |DROP2| ROT |LIST→
```

Create the new variable.

[STO]

```
3:                        133
2:                        133
1:                        151
 DUP | OVER | DUP2 |DROP2| ROT |LIST→
```

Now you can create a program that automates these steps.

Creating a Program Variable

First you'll key in the program, and then you'll store it in a variable.

Begin the program with the program delimiter.

[«]

```
2:                    133
1:                    151
«▮
 DUP  OVER  DUP2 DROP2  ROT  LIST→
```

Note that the cursor changed form and the *α* annunciator appeared, both indicating *alpha entry mode*. Pressing the key for any programmable operation writes the operation's name in the command line. Only non-programmable operations, such as pressing [◄] to erase a character, are executed.

Now key in the steps you executed before.

OVER ■ RCL
ROT   ■ PURGE
■ SWAP STO
[ENTER]

```
2:                    151
1: « OVER RCL ROT PURGE
     SWAP STO »
 DUP  OVER  DUP2 DROP2  ROT  LIST→
```

Note that the closing delimiter » was added for you.

Store the program in a variable RENAME.

['] RENAME [STO]

```
3:                    133
2:                    133
1:                    151
 DUP  OVER  DUP2 DROP2  ROT  LIST→
```

Check the USER menu.

[USER]

```
3:                    133
2:                    133
1:                    151
 RENA   B
```

Note that RENAME (in abbreviated form) appears in the USER menu.

Now you're ready to execute RENAME. You'll do it first in a round-about method, by using RCL, and then in a normal method, by using the USER menu. The difference in the methods highlights the difference between recalling and evaluating a program variable.

Recalling a Program Variable

For this example, rename the variable B to C.

Put the old name and the new name on the stack.

| ' | B | ENTER |
| ' | C | ENTER |

```
3:              151
2:              'B'
1:              'C'
RENA  B
```

Recall the program RENAME.

| ' | RENA | ■ RCL |

```
2:                  'C'
1:  « OVER RCL ROT PURGE
    SWAP STO »
RENA  B
```

For any variable, RCL simply returns the contents of the variable to the stack.

Evaluating a Program Variable

To execute a program on the stack you must *explicitly* evaluate it.

| EVAL |

```
3:              133
2:              133
1:              151
 C  RENA
```

The USER menu shows that B was renamed to C.

It wasn't necessary to recall the program to the stack for execution, but it demonstrated how RCL works for programs and how EVAL causes programs to execute. Next you'll see the easy way to execute your program.

This time you'll rename C to D. Put the old name and the new name on the stack.

| ' | C | ENTER |
| ' | D | ENTER |

```
3:              151
2:              'C'
1:              'D'
 C  RENA
```

Rename C to D.

`RENA`

```
3:                        133
2:                        133
1:                        151
   D   RENA
```

The USER menu shows that C was renamed to D. You executed the program simply by pressing one key in the USER menu.

Quoted and Unquoted Names

In the examples above you used variable names in two ways, quoted and unquoted. The quotes ⌐'⌐ are important: they distinguish the *name* of a variable from the *contents* of a variable. Here's a summary of the purposes of quoted and unquoted names.

■ Use a quoted name to represent the name itself. The quotes prevent evaluation of the name, so it goes on the stack and can be an argument to a command. In this chapter you used quoted names as arguments to STO, RCL, PURGE, and the program RENAME.

■ Use an unquoted name to evaluate the variable with that name. The unquoted name doesn't go on the stack—instead, the object stored in the variable is handled according to its type: numerical variables are returned to the stack, and programs are executed. You'll see what happens with other variable types later in this manual.

If you type in an unquoted name that isn't associated with a variable, the quoted form of the name goes on the stack.

4

Repeating Calculations

In this chapter you'll create an expression containing numerical variables and then use a calculator feature called the Solver to evaluate the expression for various values of the numerical variables.

In chapter 2 you made a calculation by keying in an expression that contained *numbers* and then evaluating the expression. In this chapter you'll create an expression by calculating on the stack, using *names* as symbolic arguments. You'll use the Solver to assign values to the variables and evaluate the expression. Each time you evaluate the expression, the calculation is made with the current values of the variables. If you change the value of one or more variables, you can simply evaluate the expression again to recalculate with the new values.

In chapter 3 you created numerical variables and a program variable. In this chapter you'll create expression variables and name variables. (Remember, any object can be stored in a variable.) You'll also learn about *directories*, which are sets of variables.

Creating an Expression

We'll repeat the resistance calculation from "Chain Calculations" in chapter 2, only this time we'll use names, rather than numbers, as arguments. Recall that the formula for the circuit is:

$$R_{total} = R_1 + \cfrac{1}{\cfrac{1}{R_2} + \cfrac{1}{R_3}}$$

Clear the stack and select the cursor menu.

■ CLEAR

```
4:
3:
2:
1:
```

If a menu is displayed, press ◄♦► to select the cursor menu.

Put the name 'R1' on the stack.

' R1 ENTER

```
4:
3:
2:
1:             'R1'
```

Note that the closing ' is added for you. You'll leave R1 on the stack until it's time to add it to the rest of the calculation.

Put the reciprocal of R2 on the stack.

' R2 ENTER ■ 1/x

```
4:
3:
2:           'R1'
1:     'INV(R2)'
```

Put the reciprocal of R3 on the stack.

' R3 ENTER ■ 1/x

```
4:
3:           'R1'
2:     'INV(R2)'
1:     'INV(R3)'
```

Add the reciprocals of R2 and R3.

+

```
4:
3:
2:           'R1'
1:  'INV(R2)+INV(R3)'
```

Take the reciprocal of the sum.

■ 1/x

```
3:
2:           'R1'
1: 'INV(INV(R2)+INV(R3)
   )'
```

Add R1 and the reciprocal.

[+]

```
3:
2:
1:  'R1+INV(INV(R2)+INV(
    R3))'
```

The resulting expression represents R_{total}.

You could key in this expression directly, taking care to add parentheses where necessary. Every expression is equivalent to a stack calculation, so you can choose the method that is easier for you.

Later in this chapter you'll store this expression in a variable, but first create a directory to group together the examples in this chapter.

Creating a Directory

A directory is a set of variables. Right now you're working in the HOME directory—a built-in directory that exists even after MEMORY RESET. In this chapter you'll create a subdirectory within HOME, and then subdirectories within that subdirectory.

Here are some concepts about directories that you'll use in this chapter.

- Only one directory can be the *current directory*; only its variables appear in the USER menu.

- If a directory A contains a directory B, then A is called the *parent directory* of B, and B is called a *subdirectory* of A.

- If you start at the current directory and find its parent directory, and then the next parent directory, and so on, you always return to HOME. This sequence of directories (in the reverse order) is called the *current path*.

You can check the current path by executing the command PATH.

Select the MEMORY menu.

■ [MEMORY]

```
2:
1:  'R1+INV(INV(R2)+INV(
    R3))'
 MEM MENU ORDER PATH HOME CRDIR
```

Check the current path.

`PATH`

```
3:
2:  'R1+INV(INV(R2)+INV...
1:              { HOME }
MEM MENU ORDER PATH HOME CRDIR
```

The list that PATH returns always begins with HOME and ends with the current directory. HOME is the starting point for all paths and, since you haven't created any other directories yet, HOME is also the current directory.

To group together all your electrical engineering problems, create a subdirectory named EE.

`'` EE `CRDIR`

```
3:
2:  'R1+INV(INV(R2)+INV...
1:              { HOME }
MEM MENU ORDER PATH HOME CRDIR
```

Switch to the EE directory.

EE `ENTER`

```
3:
2:  'R1+INV(INV(R2)+INV...
1:              { HOME }
MEM MENU ORDER PATH HOME CRDIR
```

Check the current path again.

`PATH`

```
3:  'R1+INV(INV(R2)+INV...
2:              { HOME }
1:           { HOME EE }
MEM MENU ORDER PATH HOME CRDIR
```

Now the current directory is EE. To see one effect of switching to the EE directory, display the USER menu.

`USER`

```
3:  'R1+INV(INV(R2)+INV...
2:              { HOME }
1:           { HOME EE }
```

Note that the RENAME program (created in the last chapter) doesn't appear. Only variables in the *current directory* (EE) appear in the USER menu; RENAME is in the HOME directory.

However, you could still execute RENAME by typing its name, because any variable whose directory is on the *current path* (HOME EE) can be found by name.

This is one of the benefits of directories: *If you put general-purpose programs such as RENAME in the HOME directory, you can always execute them but they don't clutter up the USER menu.*

Now you can work in the new directory EE.

Drop the two path lists from the stack.

[DROP] [DROP]

```
2:
1: 'R1+INV(INV(R2)+INV(
    R3))'
```

Store the expression in a variable named EQ1 (*equation 1*). You'll see the reason for this name later.

['] EQ1 [STO]

```
3:
2:
1:
EQ1
```

The variable EQ1 appears in the USER menu.

Let's assume that you'll use this expression for a variety of problems, each of which you want to treat independently. To do so, you can put the values for each problem in a subdirectory for that problem.

Create a subdirectory SP1 (*series-parallel 1*) for the first problem.

['] SP1 [ENTER]
CRDIR [ENTER]

```
3:
2:
1:
SP1 EQ1
```

The name of the new subdirectory appears in the USER menu. Press the menu key to switch to SP1.

SP1

```
3:
2:
1:
```

The USER menu is empty again because the current directory (SP1) is empty.

Check the current path.

PATH [ENTER]

```
3:
2:
1:          { HOME EE SP1 }
```

You can find any variable in the HOME or EE directories by name, because those directories are on the current path (HOME EE SP1), but the USER menu shows only the variables in the current directory (SP1).

Now you're ready to use the Solver with the expression EQ1.

Using the Solver To Repeat a Calculation

There are three basic steps to using the Solver with an expression.

1. Store the expression (or the name of the expression) in a variable named EQ (*equation*). The Solver requires a variable by this name.

2. Use the Solver menu to assign values to the variables.

3. Use the Solver to evaluate the expression.

4. Repeat steps 2 and 3 for other values.

Here are the steps for the present example.

Step 1. Store the name EQ1 in a variable EQ.

This step may surprise you—why store a name in a variable? Why not store the expression itself in EQ? The simplest reason is that the name EQ1 is shorter and easier to remember than the entire expression. Also, you'll see later that this makes it easy to switch back and forth between different equations.

Put the name EQ1 on the stack.

['] EQ1 [ENTER]

```
3:
2:          { HOME EE SP1 }
1:                    'EQ1'
```

If you forgot the quote ['], you got the expression itself on the stack; in this case press [DROP] to drop the expression and try again.

Select the SOLVE menu.

SOLV

```
3:
2:              { HOME EE SP1 }
1:                       'EQ1'
STEQ RCEQ SOLVR ISOL QUAD SHOW
```

Use STEQ (*Store Equation*) to store the name EQ1 in the variable EQ.

STEQ

```
3:
2:
1:              { HOME EE SP1 }
STEQ RCEQ SOLVR ISOL QUAD SHOW
```

Step 2: Assign values to the variables.

Display the Solver menu.

SOLVR

```
3:
2:
1:              { HOME EE SP1 }
R1   R2   R3  EXPR=
```

The variables in the current equation appear in the Solver menu. (If the equation contains more than six variables, pressing NEXT displays additional rows of variables.)

This menu looks different from the USER menu because it works differently: the Solver menu *stores values* in variables rather than evaluating variables.

Now you can assign values to the variables R1, R2, and R3. First store the number 8 in the variable R1.

8 R1

```
R1: 8
2:
1:              { HOME EE SP1 }
R1   R2   R3  EXPR=
```

Pressing R1 is equivalent to putting 'R1' on the stack and pressing STO. Note that the top line of the display shows the variable name and the value.

Store the number 6 in the variable R2.

6 R2

```
R2: 6
2:
1:              { HOME EE SP1 }
R1   R2   R3  EXPR=
```

Store the number 3 in the variable R3.

3 [R3]

```
┌─────────────────────────────────┐
│■R3: 3                            │
│2:                                │
│1:          { HOME EE SP1 }       │
│ R1   R2   R3  EXPR=              │
└─────────────────────────────────┘
```
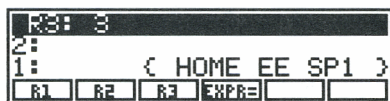
Step 3: Evaluate the expression.

The menu label EXPR= means "expression equals"—pressing it evaluates the expression.

EXPR=

```
┌─────────────────────────────────┐
│■EXPR=10                          │
│2:          { HOME EE SP1 }       │
│1:                             10 │
│ R1   R2   R3  EXPR=              │
└─────────────────────────────────┘
```
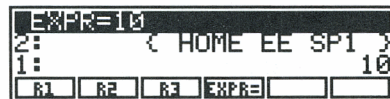
The value (10) is returned to level 1, and it appears in inverse characters in the top line of the display.

Step 4: Repeat steps 2 and 3 for other values. For example, what if R3 is 12?

Store the number 12 in the variable R3.

12 [R3]

```
┌─────────────────────────────────┐
│■R3: 12                           │
│2:          { HOME EE SP1 }       │
│1:                             10 │
│ R1   R2   R3  EXPR=              │
└─────────────────────────────────┘
```
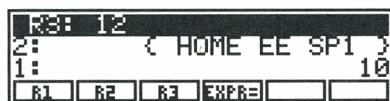
Evaluate the expression for the current values of its variables.

EXPR=

```
┌─────────────────────────────────┐
│■EXPR=12                          │
│2:                             10 │
│1:                             12 │
│ R1   R2   R3  EXPR=              │
└─────────────────────────────────┘
```
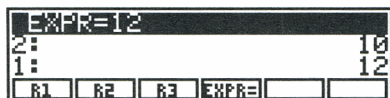
The new value (12) is returned to level 1, and it appears in inverse characters in the top line of the display.

Using a Different Set of Values

Suppose you want to work on a different problem, with different values of R1, R2, and R3, and later return to the values now assigned. You could reenter all the values each time you switch problems, but this section shows you an easier way. There are three steps:

1. Create a new directory for the new values.

2. Define the same expression to be EQ.

3. Use the Solver as before to assign values and evaluate the expression.

This process shows another benefit of directories: *Within a directory, only one variable can exist with a particular name; but any number of directories can contain a variable with a particular name.*

Step 1: Create a new directory.

Since the new directory is an alternative to SP1, call it SP2 and create it within the same parent directory (namely EE). This will be the first "branch" within your directory structure—two subdirectories (SP1 and SP2) within the same parent directory (EE).

To create a subdirectory within EE, you must make EE the current directory. (Any subdirectory you created now would be within SP1.)

Switch to the EE directory.

EE [ENTER]

```
No Current Equation
2:                    10
1:                    12
STEQ RCEQ SOLVR ISOL QUAD SHOW
```

The calculator beeps, displays No Current Equation, and activates the SOLVE menu. This occurs because there is no current equation 'EQ' in the EE directory.

Create a directory SP2.

['] SP2
[■ MEMORY] CRDIR

```
3:         { HOME EE SP1 }
2:                    10
1:                    12
MEM MENU ORDER PATH HOME CRDIR
```

Switch to the SP2 directory.

SP2 [ENTER]

```
3:              { HOME EE SP1 }
2:                           10
1:                           12
MEM |MENU|ORDER| PATH | HOME |CRDIR
```

Check the current path.

PATH

```
3:                           10
2:                           12
1:              { HOME EE SP2 }
MEM |MENU|ORDER| PATH | HOME |CRDIR
```

Note that HOME and EE are in the current path, as they were when SP1 was the current directory, but SP1 doesn't appear now. As a result, you can still find the variables in HOME (such as RENAME) and in EE (such as EQ1), but not the variables in SP1 (EQ, R1, R2, and R3); now you can create new variables R1, R2, and R3.

Step 2: Define the same expression to be EQ.

As before, use STEQ to store the name EQ1 in the variable EQ.

['] EQ1 [SOLV] STEQ

```
3:                           10
2:                           12
1:              { HOME EE SP2 }
STEQ |RCEQ|SOLVR| ISOL | QUAD |SHOW
```

Step 3: Use the Solver as before to assign values and evaluate the expression. Suppose the values for the new problem are

$$R1 = 11, \quad R2 = 21, \quad R3 = 7$$

Select the Solver menu.

SOLVR

```
3:                           10
2:                           12
1:              { HOME EE SP2 }
 R1 || R2 || R3 ||EXPR=||   ||
```

Assign the values.

11 [R1]
21 [R2]
7 [R3]

```
 R3: 7
2:                           12
1:              { HOME EE SP2 }
 R1 || R2 || R3 ||EXPR=||   ||
```

Evaluate the expression.

`EXPR=`

```
█EXPR=16.25█████████████
2:            { HOME EE  SP2 }
1:                       16.25
 [ R1 ][ R2 ][ R3 ][EXPR=][  ][  ]
```

To return to the previous problem, you would execute EE (to switch to the EE directory), execute SP1 (to switch to the SP1 directory), and press `SOLV` `SOLVR` (to activate the Solver menu); all the variable values would be the same as when you left SP1.

Using a Different Expression

Now that you have two sets of values to use with the expression EQ1, try creating a second expression EQ2 that you can use with either set of values. There are two basic steps:

1. Switch to the EE directory, create the new expression, and store the new expression in a variable EQ2.

2. Switch to the SP1 or SP2 directory, change the value of EQ from 'EQ1' to 'EQ2', and use the Solver to evaluate the expression.

Step 1: Switch to the EE directory, create the new expression, and store the new expression in a variable EQ2.

Switch to the EE directory.

EE `ENTER`

```
No Current Equation
2:            { HOME EE  SP2 }
1:                       16.25
[STEQ][RCEQ][SOLVR][ISOL][QUAD][SHOW]
```

Create the new expression. In this example, EQ2 will be an edited copy of the expression EQ1.

Return the expression stored in EQ1 to the stack.

USER EQ1

```
2:                          16.25
1: 'R1+INV(INV(R2)+INV(
   R3))'
 SP2  SP1  EQ1
```

Return the expression to the command line.

■EDIT

```
1: 'R1+INV(INV(R2)+INV(
   R3))'
'R1+INV(INV(R2)+INV(
R3))'
```

The expression in level 1 appears in inverse characters to warn you that it will be replaced by the contents of the command line. The alpha annunciator *α* appears, indicating that alpha entry mode is active.

Now edit the expression to represent the formula:

$$R_{total} = R_1 + \cfrac{1}{\cfrac{1}{R_2} + \cfrac{1}{R_3} + \cfrac{1}{R_3}}$$

Move the cursor to the lower row of the command line. (The operations for moving the cursor are on the *cursor menu*—the labels printed in white just above the menu keys.)

▼

```
1: 'R1+INV(INV(R2)+INV(
   R3))'
'R1+INV(INV(R2)+INV(
■3))'
```

The cursor menu is active whenever the command line exists and no menu is displayed. You can turn the cursor menu on and off by pressing ◄►. Pressing ■EDIT automatically turns on the cursor menu.

Move the cursor just past the term for R3.

► ► ►

```
1: 'R1+INV(INV(R2)+INV(
   R3))'
'R1+INV(INV(R2)+INV(
R3)■'
```

Select Insert mode.

INS

```
1: 'R1+INV(INV(R2)+INV(
     R3))'
'R1+INV(INV(R2)+INV(
R3)◄'
```

The shape of the cursor changes to an arrow, indicating that text will be inserted to the left of the character at the cursor position. (Pressing INS a second time returns to replace mode, where text replaces the character at the cursor position.)

Key in the second term for R3.

+ ■ 1/x (R3

```
1: 'R1+INV(INV(R2)+INV(
     R3))'
'R1+INV(INV(R2)+INV(
R3) + INV (R3◄'
```

Replace the expression in level 1 by the edited expression in the command line.

ENTER

```
2:                    16.25
1: 'R1+INV(INV(R2)+INV(
     R3)+INV(R3))'
| SP2 | SP1 | EQ1 |     |     |
```

Store the new expression in a variable EQ2.

' EQ2 STO

```
3:                       12
2:         { HOME EE SP2 }
1:                    16.25
| EQ2 | SP2 | SP1 | EQ1 |     |
```

Step 2: Switch to the SP1 or SP2 directory, change the definition of EQ from EQ1 to EQ2, and use the Solver to evaluate the expression.

For this example, use the values in SP1 with the new expression.

Switch to the SP1 directory.

SP1

```
3:                       12
2:         { HOME EE SP2 }
1:                    16.25
| R3 | R2 | R1 | EQ |     |
```

Change the definition of EQ from EQ1 to EQ2.

⌐'⌐ EQ2 [SOLV] `STEQ`

```
3:                          12
2:          { HOME EE SP2 }
1:                      16.25
STEQ RCEQ SOLVR ISOL QUAD SHOW
```

Evaluate the expression EQ2 with the values from SP1.

`SOLVR` [EXPR=]

```
EXPR=11
2:                      16.25
1:                         11
 R1    R2    R3   EXPR=
```

To evaluate EQ2 with the values from SP2, you could execute EE (to switch directories back to EE) and then repeat step 2 above, substituting SP2 for SP1.

Returning to HOME

Assuming you're done for now with your electrical engineering problems, you can return to the HOME directory. Since HOME is a built-in directory, its name is included in the MEMORY menu.

Switch to the HOME directory.

■ [MEMORY] `HOME`

```
3:          { HOME EE SP2 }
2:                      16.25
1:                         11
MEM MENU ORDER PATH HOME CRDIR
```

Check the USER menu.

[USER]

```
3:          { HOME EE SP2 }
2:                      16.25
1:                         11
 EE    D   RENA
```

The menu label `EE` is the only sign of everything you created in this chapter—EQ1, EQ2, the subdirectories SP1 and SP2, and all the variables in them. This is a major advantage of directories: *Viewed from its parent directory, an entire directory—its variables and its own subdirectories—appear simply as the name of the directory.*

Summary

Here's the overall strategy you've followed in this chapter.

- Create a directory for each set of related problems.
- Store each expression needed for the problems in a variable.
- Create a subdirectory for the specific values in each problem.
- Use the Solver with any combination of expression and values.

5

Real-Number Functions

This chapter introduces the TRIG, LOGS, and REAL menus. The TRIG menu contains trigonometric functions and commands dealing with angular measurement. The LOGS menu contains logarithmic, exponential, and hyperbolic functions. The REAL menu contains additional commands for real numbers.

All commands in these menus are described briefly in appendix D, "Menu Map." For complete descriptions, refer to "TRIG," "LOGS," or "REAL" in the Reference Manual.

Trigonometric Functions

This section shows how to select the current angle mode, calculate with π, and convert angular measure.

Selecting Angle Mode

The calculator can interpret angular arguments and results as degrees ($\frac{1}{360}$ of a circle) or as radians ($\frac{1}{2}\pi$ of a circle). The default choice is Degrees angle mode. For the examples in this section, switch to Radians angle mode.

Clear the stack and select the MODE menu.

■ CLEAR ■ MODE

```
3:
2:
1:
STD∎ | FIX | SCI | ENG | DEG∎ | RAD
```

The two right-most menu labels, DEG (*degrees*) and RAD (*radians*), represent your choices of angle mode. Note that the DEG label shows a small square, indicating that the current angle mode is Degrees.

Select Radians angle mode.

RAD

```
3:
2:
1:
STD■ FIX  SCI  ENG  DEG  RAD■
```
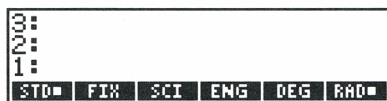
The Radians annunciator **(2π)** appears and the menu labels change. (Most illustrations in this manual don't show the annunciators. To locate the **(2π)** annunciator, see the illustration on page 27.)

Display the first row of the TRIG menu.

TRIG

```
3:
2:
1:
SIN  ASIN  COS  ACOS  TAN  ATAN
```
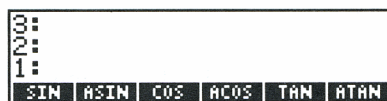
These are one-number functions, acting on the number in level 1. For real numbers, the angle mode affects how SIN (*sine*), COS (*cosine*), and TAN (*tangent*) interpret their arguments, and how ASIN (*arc sine*), ACOS (*arc cosine*), and ATAN (*arc tangent*) express their results.

You'll use the SIN function in the discussion of π that comes next.

Using π

The transcendental number π can't be represented exactly in a finite decimal form. In general, the calculator's 12-digit approximation (3.14159265359) yields results accurate to 12 digits, which is sufficient for most applications.

The HP-28S also offers a symbolic constant π that represents π exactly. In radians angle mode, the functions SIN, COS, and TAN recognize the symbolic constant π and produce an exact result. The functions SIN and COS also recognize π/2.

For other functions, the symbolic constant π produces an expression containing π. If you force a real-number result, the calculator uses the 12-digit approximation.

To demonstrate the difference between 3.14159265359 and π, calculate the sine of each.

Put 'π' in level 1.

■ π [ENTER]

```
3:
2:
1:                        'π'
 SIN  ASIN   COS   ACOS   TAN   ATAN
```

Although this object looks like a name, it's actually an expression with a single term, the symbolic constant π.

Force a real-number result using →NUM (to number)

■ [→NUM]

```
3:
2:
1:            3.14159265359
 SIN  ASIN   COS   ACOS   TAN   ATAN
```

The 12-digit approximation to π (3.14159265359) is returned to level 1.

Calculate the sine of the approximation to π.

SIN

```
3:
2:
1:      -2.06761537357E-13
 SIN  ASIN   COS   ACOS   TAN   ATAN
```

The result ($-2.06761537357 \times 10^{-13}$) isn't exactly 0 because the argument (3.14159265359) isn't exactly π.

Now calculate the sine of π.

■ π   SIN

```
3:
2:      -2.06761537357E-13
1:                        0
 SIN  ASIN   COS   ACOS   TAN   ATAN
```

The SIN function recognizes the symbolic constant π and returns the exact result (0).

Converting Angular Measure

The TRIG menu contains commands that convert an angle from one system of measurement to another. These commands are on the third row of the TRIG menu. Take a quick look at the second row before continuing to the third.

Display the second row of the TRIG menu.

[NEXT]

```
3:
2:        -2.06761537357E-13
1:                          0
 P→R  R→P  R→C  C→R  ARG
```

These commands deal with complex numbers and are duplicated in the COMPLEX menu. Complex numbers are described in the next chapter.

Display the third row of the TRIG menu.

[NEXT]

```
3:
2:        -2.06761537357E-13
1:                          0
 →HMS HMS→ HMS+ HMS-  D→R  R→D
```

You'll use the commands HMS→ and D→R to convert an angle expressed in degrees, minutes, and seconds to an angle expressed in radians.

The four HMS (*hours-minutes-seconds*) commands enable you to calculate with numbers whose fractional parts are expressed as minutes and seconds. Such numbers must have the following special format, called the HMS format:

h.MMSSs

where *h* represents hours (or degrees), *MM* represents minutes, *SS* represents seconds, and *s* represents decimal fraction of seconds. *MM* and *SS* each represent two digits; *h* and *s* each represent any number of digits.

The commands →HMS (*decimal-to-HMS*) and HMS→ (*HMS-to-decimal*) convert a real number between the normal decimal format and the special HMS format. The commands HMS+ (*HMS plus*) and HMS− (*HMS minus*) add and subtract numbers in HMS format, with the result also in HMS format.

For example, convert 141° 26′ 15″ to decimal degrees.

Enter the number in HMS format.

141.2615

```
2:        -2.06761537357E-13
1:                          0
141.2615□
+HMS HMS+ HMS+ HMS-   D+R    R+D
```

Convert the number from HMS format to decimal degrees.

HMS+

```
3:        -2.06761537357E-13
2:                          0
1:                   141.4375
+HMS HMS+ HMS+ HMS-   D+R    R+D
```

The other two functions on this menu row, D→R (*degrees-to-radians*) and R→D (*radians-to-degrees*) convert a real number between degrees angular measure and radians angular measure.

Convert the number in level 1 from degrees to radians.

D→R

```
3:        -2.06761537357E-13
2:                          0
1:             2.46855006079
+HMS HMS+ HMS+ HMS-   D+R    R+D
```

Altogether, you've calculated:

141° 26′ 15″ = 141.4375° = 2.46855006079 radians

Logarithmic, Exponential, and Hyperbolic Functions

The LOGS menu contains logarithmic and exponential functions, both common and natural, and hyperbolic functions. For a detailed description of these functions, refer to "LOGS" in the Reference Manual.

Display the first row of the LOGS menu.

■ LOGS

```
3:        -2.06761537357E-13
2:                          0
1:             2.46855006079
LOG  ALOG   LN    EXP  LNP1 EXPM
```

The functions LOG (*common logarithm*) and ALOG (*common antiloga-rithm*) compute logarithms and exponentials to base 10. The functions LN (*natural logarithm*) and EXP (*natural exponential*) calculate loga-rithms and exponentials to base e. (e is a transcendental number approximately equal to 2.71828182846.)

For an argument x, the function LNP1 (*ln plus 1*) computes ln $(x + 1)$, and the function EXPM (*exp minus 1*) computes (exp x) − 1. For argu-ments close to 0, each of these functions provides greater accuracy than the corresponding sequence of functions. (An example using LNP1 appears in "Time Value of Money" on page 103.)

Display the second row of the LOGS menu.

NEXT

```
3:         -2.06761537357E-13
2:                          0
1:             2.46855006079
 SINH ASINH COSH ACOSH TANH ATANH
```

These are the hyperbolic functions and their inverses: SINH (*hyper-bolic sine*) and ASINH (*inverse hyperbolic sine*), COSH (*hyperbolic cosine*) and ACOSH (*inverse hyperbolic cosine*), and TANH (*hyperbolic tangent*) and ATANH (*inverse hyperbolic tangent*). These functions are derived from e^x, the natural exponential function. All are one-number functions that act on the number in level 1.

Other Real Functions

The REAL menu contains functions that apply primarily to real numbers.

Select the REAL menu.

■REAL

```
3:         -2.06761537357E-13
2:                          0
1:             2.46855006079
 NEG  FACT  RAND  RDZ  MAXR  MINR
```

The function NEG (*negate*) returns −x for an argument x. The func-tion FACT (*factorial*) returns $n!$ for a positive integer n or the gamma function $\Gamma(x + 1)$ for a non-integer argument x. The command RAND (*random number*) returns a random number calculated from a seed specified by RDZ (*randomize*).

The functions MAXR (*maximum real*) and MINR (*minimum real*) return symbolic constants for the largest and smallest positive real numbers representable on the HP-28S. (To force a numerical result for a symbolic constant, see "Using π" on page 74.)

This section shows you how to use the function NEG. For convenience, you can execute NEG by pressing [CHS] (*change sign*) if no command line is present. To enter the NEG command in the command line—for example, when you're keying in a program—press `NEG` or [N] [E] [G].

Now negate the number in level 1 twice, once by pressing [CHS] and once by pressing `NEG`.

Negate the number in level 1.

[CHS]

```
3:       -2.06761537357E-13
2:                         0
1:          -2.46855006079
 NEG  FACT  RAND  RDZ  MAXR MINR
```

Negate the number a second time.

`NEG`

```
3:       -2.06761537357E-13
2:                         0
1:           2.46855006079
 NEG  FACT  RAND  RDZ  MAXR MINR
```

Defining New Functions

You can create program variables that work like the built-in functions—you can even use them in expressions. Such program variables, called *user functions*, must fulfill two requirements:

- They must explicitly indicate their arguments.
- They must return exactly one result.

For example, you can define a function COT for the cotangent function, where cot $x = 1/\tan x$.

Begin the program.

| « |

```
2:                        0
1:           2.46855006079
«█
 NEG  FACT  RAND  RD2  MAXR  MINR
```

Indicate the argument.

█ → LC x LC

```
2:                        0
1:           2.46855006079
« → x█
 NEG  FACT  RAND  RD2  MAXR  MINR
```

The right arrow indicates that the following name represents a *local variable*, which will exist only within this program.

It's useful to follow some convention to distinguish your local variables from your ordinary or "global" variables. This manual uses lower-case letters to distinguish local variables. (Pressing LC once switches to lower case; pressing LC a second time switches back to upper case.)

Define the function.

' █ 1/x (I TAN (I LC x

```
2:                        0
1:           2.46855006079
« → x' INV (TAN(x█
 NEG  FACT  RAND  RD2  MAXR  MINR
```

Enter the program.

ENTER

```
2:           2.46855006079
1: « → x 'INV(TAN(x))'
    »
 NEG  FACT  RAND  RD2  MAXR  MINR
```

The closing parentheses and delimiters are added for you.

This program means: take an argument from the stack (in RPN syntax) or from the expression (in algebraic syntax) and call it *x*; then evaluate the expression 1/tan *x*, using the local definition of *x*.

Store the program in a variable COT.

['] COT [STO]

```
3:        -2.06761537357E-13
2:                          0
1:              2.46855006079
 NEG  FACT  RAND  RDZ  MAXR  MINR
```

Now you can use COT in either RPN or algebraic syntax, just like the built-in trigonometric functions.

Calculate cot 45° using RPN.

■ [MODE] DEG
45 [USER] COT

```
3:                          0
2:              2.46855006079
1:                          1
 COT   EE    D   RENA
```

Calculate cot −45° using algebraic syntax.

['] COT [(] −45 [ENTER]

```
3:              2.46855006079
2:                          1
1:               'COT(-45)'
 COT   EE    D   RENA
```

Evaluate the expression.

[EVAL]

```
3:              2.46855006079
2:                          1
1:                         -1
 COT   EE    D   RENA
```

6

Complex-Number Functions

The HP-28S includes an object type that represents complex numbers. For example, the complex number $z = 3 + 4i$ is represented by the object $(3,4)$. Because each complex number is a single object, you can calculate with complex numbers as easily as real numbers.

The pair of real numbers in a complex number can represent the co-ordinates of a point in a plane. For example, the HP-28S uses complex numbers to represent plotting coordinates. The second section in this chapter describes two coordinate systems, rectangular and polar, and shows how to convert a point from one system to the other.

Using Complex Numbers

Most functions that work with real numbers work the same way with complex numbers. For example, you do arithmetic with complex numbers just as you do real numbers—put the numbers on the stack and execute the function. Try calculating the following:

$$((9 + 2i) + (-4 + 3i)) \times (6 + i)$$

Clear the stack and enter $9 + 2i$.

■ CLEAR
◖ 9 , 2 ENTER

```
3:
2:
1:                        (9,2)
 COT │ EE │ D │ RENA │    │
```

Add $-4 + 3i$. (Remember to press ④ CHS to enter -4.)

(| -4 ⨟ 3 +

```
3:
2:
1:            (5,5)
 COT | EE | D |RENA|    |
```

Multiply by $6 + i$.

(| 6 ⨟ 1 ×

```
3:
2:
1:          (25,35)
 COT | EE | D |RENA|    |
```

Sometimes a real-number argument can produce a complex-number result.

Calculate $\sqrt{-4}$.

-4 ■ √x̄

```
3:
2:          (25,35)
1:            (0,2)
 COT | EE | D |RENA|    |
```

Calculate arcsin 2.

2 TRIG ASIN

```
2:            (0,2)
1: (1.57079632679,
     -1.31695789692)
 SIN |ASIN| COS |ACOS| TAN |ATAN|
```

Functions specifically for complex numbers are in the COMPLEX menu.

Select the COMPLEX menu.

■ COMPLX

```
2:            (0,2)
1: (1.57079632679,
     -1.31695789692)
 R→C | C→R | RE | IM |CONJ|SIGN|
```

All commands in the COMPLEX menu are described briefly in appendix D, "Menu Map." For complete descriptions, refer to "COMPLEX" in the Reference Manual.

■ R→C (*real-to-complex*) converts two real numbers x and y to one complex number (x, y).

■ C→R (*complex-to-real*) converts one complex number (x, y) to two real numbers x and y.

■ RE (*real part*) returns x for a complex argument (x, y).

- IM (*imaginary part*) returns y for a complex argument (x, y).
- CONJ (*conjugate*) returns $(x, -y)$ for a complex argument (x, y).
- SIGN returns $(x/\sqrt{x^2 + y^2}, y/\sqrt{x^2 + y^2})$ for a complex argument (x, y).

Display the next row of the COMPLEX menu.

| NEXT |

```
2:                    (0,2)
1: (1.57079632679,
   -1.31695789692)
 R→P  P→R  ABS  NEG  ARG
```
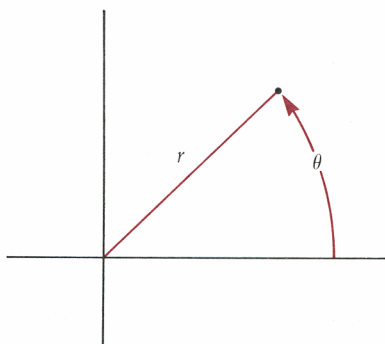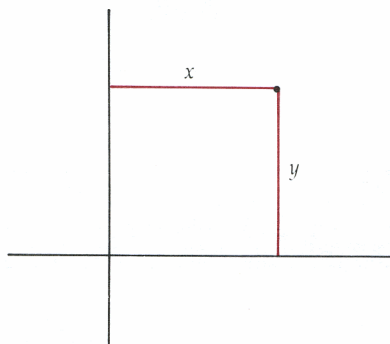
These functions (except NEG) relate to complex numbers in polar coordinates.

Using Polar Coordinates

A point in a plane can be described by two different coordinate systems. The illustration below shows one point described two ways, in rectangular notation (x, y) and in polar notation (r, θ).

- R→P (*rectangular-to-polar*) converts a complex number in rectangular notation (*x*, *y*) to polar notation (*r*, *θ*).

- P→R (*polar-to-rectangular*) converts a complex number in polar notation (*r*, *θ*) to rectangular notation (*x*, *y*).

- ABS (*absolute value*) returns *r* for a complex argument (*x*, *y*).

- NEG returns (−*x*, −*y*) for a complex argument (*x*, *y*).

- ARG returns *θ* for a complex argument (*x*, *y*).

Note that only P→R interprets a complex number as polar coordinates; all other functions—arithmetic, trigonometric, logarithmic, hyperbolic, and so on—interpret a complex number as rectangular coordinates. Remember this important rule: *Any complex number in polar coordinates must be converted to rectangular coordinates before you can use it in a calculation.*

As an example of arithmetic with polar coordinates, suppose you travel 2 miles at a bearing of 36°, then 3 miles at a bearing of 65°. What is the resulting distance and bearing, calculated to two decimal places?

Select Degrees angle mode and FIX 2 number display.

■ MODE   DEG 2 FIX

```
3:          (25.00,35.00)
2:            (0.00,2.00)
1:           (1.57,-1.32)
 STD | FIX■| SCI | ENG |DEG■| RAD
```

Enter the first distance and bearing.

((2 (,) 36

```
2:            (0.00,2.00)
1:           (1.57,-1.32)
(2,36□
 STD | FIX■| SCI | ENG |DEG■| RAD
```

Convert to rectangular coordinates.

■ COMPLX   NEXT   P→R

```
3:            (0.00,2.00)
2:           (1.57,-1.32)
1:            (1.62,1.18)
 R→P | P→R | ABS | NEG | ARG |
```

Enter the second distance and bearing.

((3 (,) 65

```
2:           (1.57,-1.32)
1:            (1.62,1.18)
(3,65□
 R→P | P→R | ABS | NEG | ARG |
```

Convert to rectangular coordinates.

P→R

```
3:              (1.57,-1.32)
2:               (1.62,1.18)
1:               (1.27,2.72)
 R→P   P→R   ABS   NEG   ARG
```

Add the rectangular coordinates.

+

```
3:               (0.00,2.00)
2:              (1.57,-1.32)
1:               (2.89,3.89)
 R→P   P→R   ABS   NEG   ARG
```

Convert to polar coordinates.

R→P

```
3:               (0.00,2.00)
2:              (1.57,-1.32)
1:              (4.85,53.46)
 R→P   P→R   ABS   NEG   ARG
```

The resulting distance is 4.85 miles, and the resulting bearing is 53.46°.

A User Function for Polar Addition

Here's a simple program PSUM (*polar sum*) to automate the process you did manually in the previous section.

Begin the program.

«

```
2:              (1.57,-1.32)
1:              (4.85,53.46)
«
 R→P   P→R   ABS   NEG   ARG
```

Indicate the arguments. (Use a space to separate the two arguments.)

■ → LC x SPACE y

```
2:              (1.57,-1.32)
1:              (4.85,53.46)
« → x y
 R→P   P→R   ABS   NEG   ARG
```

The right arrow indicates that the following names are *local variables*, which will exist only within this program.

Define the function.

⌐'⌐ `R→P` ⌐(⌐ `P→R` ⌐(⌐ x ■⌐)⌐ ⌐+⌐
`P→R` ⌐(⌐ y ⌐ENTER⌐

```
2:              (4.85,53.46)
1: « → x y 'R→P(P→R(x)+
      P→R(y))' »
 R→P | P→R | ABS | NEG | ARG
```

The closing parentheses and delimiters are added for you.

This program means: take two arguments from the stack (in RPN syntax) or from the expression (in algebraic syntax) and call them x and y; then calculate the polar coordinates of the sum of the rectangular coordinates of x and y.

Store the program in a variable PSUM.

⌐'⌐ PSUM ⌐STO⌐

```
3:              (0.00,2.00)
2:              (1.57,-1.32)
1:              (4.85,53.46)
 R→P | P→R | ABS | NEG | ARG
```

Now use PSUM to repeat the previous calculation, once in RPN syntax and once in algebraic syntax.

Enter the first distance and bearing.

⌐(⌐ 2 ⌐,⌐ 36 ⌐ENTER⌐

```
3:              (1.57,-1.32)
2:              (4.85,53.46)
1:              (2.00,36.00)
 R→P | P→R | ABS | NEG | ARG
```

Enter the second distance and bearing.

⌐(⌐ 3 ⌐,⌐ 65

```
2:              (4.85,53.46)
1:              (2.00,36.00)
(3,65□
 R→P | P→R | ABS | NEG | ARG
```

Execute PSUM.

⌐USER⌐ `PSUM`

```
3:              (1.57,-1.32)
2:              (4.85,53.46)
1:              (4.85,53.46)
 PSUM| COT | EE | D |RENA
```

The result matches the previous answer.

Now try algebraic syntax.

⌐'⌐ PSUM ⌐(⌐ ⌐(⌐ 2 ⌐,⌐ 36 ■⌐)⌐ ⌐,⌐ ⌐(⌐ 3   2:   (4.85,53.46)
⌐,⌐ 65 [ENTER]   1: ¦PSUM((2,36),(3,65))
                                             ┌─────┬────┬────┬───┬─────┬─┐
                                             │PSUM │ COT│ EE │ D │RENA │ │
                                             └─────┴────┴────┴───┴─────┴─┘

The outer parentheses and the center comma define the arguments to
PSUM; the other parentheses and commas are part of complex-num-
ber syntax. Don't forget that you need *two* sets of parentheses when
using a complex number as an argument in algebraic syntax.

Evaluate the expression.

[EVAL]   3:   (4.85,53.46)
                                             2:                    (4.85,53.46)
                                             1:                    (4.85,53.46)
                                             ┌─────┬────┬────┬───┬─────┬─┐
                                             │PSUM │ COT│ EE │ D │RENA │ │
                                             └─────┴────┴────┴───┴─────┴─┘

7

Plotting

This chapter introduces plotting on the HP-28S. Plotting is helpful in itself, giving a visual understanding of how an expression or equation behaves. In addition, plotting makes it easy to estimate the roots, maxima, or minima of an expression. The next chapter, "The Solver," shows how to use the Solver to turn estimates into precise numbers.

In this chapter you'll learn how to use some of the commands in the PLOT menu. All commands in the PLOT menu are described briefly in appendix D, "Menu Map." For complete descriptions, refer to "PLOT" in the Reference Manual.

For the first example you'll plot $\sin x$ in Radians angle mode, but first there are preliminaries to make sure your display will match the illustrations.

Plotting uses a variable named PPAR to store a list of plotting parameters. Purge any existing PPAR to ensure that the next plot uses the default plotting parameters.

Clear the stack and select the PLOT menu.

■ CLEAR
■ PLOT

```
3:
2:
1:
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Display the second row of the PLOT menu.

NEXT

```
3:
2:
1:
PPAR RES AXES CENTR *W *H
```

Purge any existing PPAR.

`'` `PPAR` ■`PURGE`

```
3:
2:
1:
PPAR | RES | AXES |CENTR| *W | *H
```

Select Radians angle mode and STD number display mode.

■`MODE` `RAD` `STD`

```
3:
2:
1:
STD■| FIX | SCI | ENG | DEG |RAD■
```

Now enter the expression.

`'` `TRIG` `SIN` X `ENTER`

```
3:
2:
1:          'SIN(X)'
SIN | ASIN | COS | ACOS | TAN | ATAN
```

Store the expression as the *current equation*—a normal variable with the special name EQ. (This is the same convention you followed with the Solver in chapter 4.)

■`PLOT` `STEQ`

```
3:
2:
1:
STEQ |RCEQ |PMIN |PMAX |INDEP|DRAW
```

Pressing `STEQ` is equivalent to pressing `'` EQ `STO`.

Plot the expression.

`DRAW`
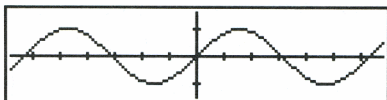
Wait for the ((●)) annunciator to disappear, indicating that the plot is complete.

The horizontal line is the axis for the independent variable (x in this example), and the vertical line is the axis for the dependent variable (the value of the expression sin x). The ticks on both axes mark intervals of length 1.

Printing a Plot

If you have an HP 82240A printer, you can print an image of the plot you just made as follows.

1. Position the printer according to the instructions in the printer manual.

2. Press and hold [ON].

3. Press [L] (the key with "PRINT" written above it).

4. Release [ON].

These keystrokes are the keyboard equivalent of the command PRLCD (*print LCD*, found on the first row of the PRINT menu). You can use these keystrokes to print the display at practically any time, without disturbing calculator operation.

If you write a program to plot an expression and print the result, use the following sequence of commands:

$$\ldots \text{CLLCD DRAW PRLCD} \ldots$$

Returning to the present example, now restore the normal display of the stack.

[ON]

```
3:
2:
1:
STEQ  RCEQ  PMIN  PMAX  INDEP  DRAW
```

Changing the Scale of the Plot

In general, plotting an expression doesn't produce such tidy results the first time. When you're plotting an unfamiliar expression you may need to adjust the plotting region—defined by the plotting parameters—to show the relevant characteristics of the expression.

If you know beforehand the region that you want to plot, you can directly change the plotting parameters in PPAR. (PPAR is described in detail in "PLOT" in the Reference Manual.) More often you need to experiment to find the desired plotting region. This manual shows you how to use commands in the PLOT menu to "home in" on the desired plot.

For the second example, you'll plot the expression $x^3 - x^2 - x + 3$.

Put the expression in level 1.

⌐'⌐ X ■ ⌐^⌐ 3 ⌐−⌐ X ■ ⌐^⌐ 2 ⌐−⌐ X ⌐+⌐ 3
⌐ENTER⌐

```
3:
2:
1:         'X^3-X^2-X+3'
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Store the expression as the current equation.

STEQ

```
3:
2:
1:
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Plot the expression.

DRAW
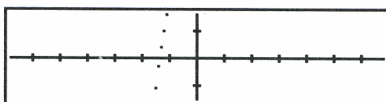
The horizontal line is the axis for x, and the vertical line is the axis for the value of the expression $x^3 - x^2 - x + 3$.

This plot shows a *zero* of the expression—a value of X for which the value of the expression is zero. The zero is located where the graph of the expression crosses the X axis. In the next chapter we'll use the Solver to find a precise number for this zero.

To show more of the graph, expand the vertical scale and plot again.

Restore the normal display of the stack.
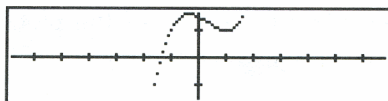
⌐ON⌐

```
3:
2:
1:
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Expand the height by a scaling factor of 2, using the `*H` (*times height*) operation on the next menu row.

NEXT 2 `*H`

```
3:
2:
1:
PPAR  RES  AXES CENTR  *W    *H
```

Plot again with the new plot parameters.

■ PREV DRAW

The ticks on the horizontal axis still mark off intervals of length 1, but now the tick marks on the vertical axis mark off intervals of length 2.

Next you'll translate the plot, moving the interesting part to the center of the display.

Translating the Plot

After each plot the calculator leaves cross hairs in the center of the display. (You can't see the cross hairs when the axes are in the center of the display.) You can use the cross hairs to *digitize* any point on the display, returning the coordinates of the point to the stack. We'll digitize the point we want to be the center of the next plot and use it to adjust the plotting parameters.

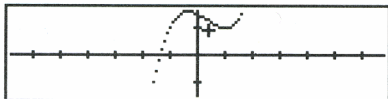Move the cross hairs to the indicated position.

▶ (press four times)
▲ (press nine times)

Digitize the point.

INS
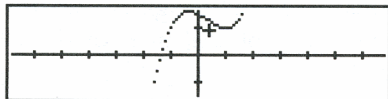
Return to the stack display.

[ON]

```
3:
2:
1:                    (.4,1.8)
STEQ RCEQ PMIN PMAX INDEP DRAW
```

The coordinates of the digitized point, represented by a complex number, are in level 1.

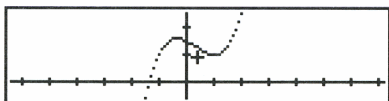Redefine the center of the plot, using CENTR on the next menu row.

[NEXT] CENTR

```
3:
2:
1:
PPAR  RES  AXES CENTR  *W    *H
```

The coordinates are taken from the stack and used to adjust the plot parameters. Unlike *H , CENTR doesn't change the scale.

Try another plot.

■[PREV] DRAW

Now zoom in on an interesting part of the plot. You could use *H again, using a *fractional* scaling factor. (For example, a scaling factor of .5 would return the vertical scale to its original value.) But there's a more flexible way to zoom in on a plot.
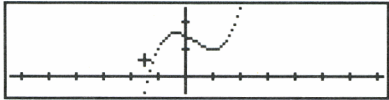
Redefining the Corners of the Plot

This time you'll digitize two points, one for the lower-left corner of the new plot and one for the upper-right corner, to zoom in on the plot.

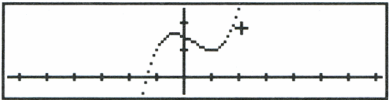Move the cross hairs to the desired lower-left corner.
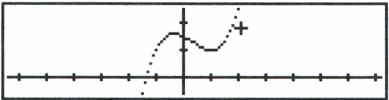
Digitize the point.

`INS`
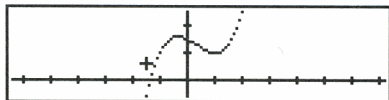
Move the cross hairs to the desired upper-right corner.

Digitize the point.

`INS`

Return to the stack display.

`ON`

```
3:
2:              (-1.5,1.2)
1:               (2.1,3.6)
STEQ RCEQ PMIN PMAX INDEP DRAW
```

The coordinates of the lower-left corner, represented by a complex number, are in level 2. The coordinates of the upper-right corner are in level 1. (Your coordinates may differ slightly from the illustration.)

Redefine the upper-right corner of the plot, using PMAX (*plot maxima*).

PMAX

```
3:
2:
1:              (-1.5,1.2)
STEQ RCEQ PMIN PMAX INDEP DRAW
```

The coordinates are taken from the stack and used to adjust the plotting parameters.

Redefine the lower-left corner of the plot, using PMIN (*plot minima*).

PMIN

```
3:
2:
1:
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Try another plot.

DRAW
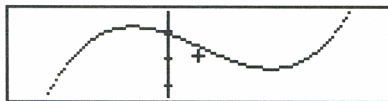
Since you changed the height and width of the plot, both the vertical and horizontal scales are changed.

The plot shows two *extrema* in the expression's graph—a local maximum and a local minimum. In the next chapter you'll use the Solver to find a precise value for the minimum. To avoid repeating all these steps to generate our current plotting parameters, store the current value of PPAR in a variable with a different name. To recreate this plot in the next chapter, you'll restore PPAR to its current value.

Return to the stack display.

ON

Put the current contents of PPAR on the stack.

NEXT   PPAR

For information about the plotting parameters and for details about plotting in general, see "PLOT" in the Reference Manual.

Create a variable PPAR1 that contains the current plotting parameters.

[']   PPAR   1 [STO]

Now you're ready to use the Solver to find precise numbers for the zero and local minimum of the expression.

Plotting Equations

The examples in this chapter were both expressions, but the same rules and techniques work for plotting equations. When the variable EQ contains an equation, DRAW plots *each side* of the equation as an expression. You can find a root of the equation by finding where the two graphs cross, because that is where the two sides of the equation have equal values.

The Solver

This chapter describes how to find a zero and a minimum of the expression you plotted in the previous chapter. Work through the steps in the previous chapter if you haven't done so already, because you'll need some of the results from that chapter.

For a complete description of the Solver, refer to "SOLVE" in the Reference Manual.

Finding a Zero of an Expression

The following example assumes that the expression $x^3 - x^2 - x + 3$ is still the current equation and that you've created the variable PPAR1, as described in the previous chapter. You'll plot the expression again, digitize an estimate for a zero of the expression, and then use the Solver to find a more accurate value for the zero.

Before starting these examples, clear the stack, select Radians angle mode, and select FIX 2 number display mode.

■ CLEAR						
■ MODE	RAD					
2 FIX						

3:					
2:					
1:					
STD	FIX■	SCI	ENG	DEG	RAD■

Purge the existing PPAR to ensure that the next plot uses the default plotting parameters.

| ■ PLOT | NEXT |
| ' PPAR | ■ PURGE |

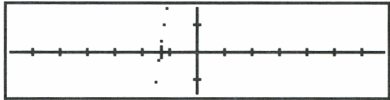| 3: | | | | | |
2:					
1:					
PPAR	RES	AXES	CENTR	∗W	∗H

Now plot the expression.

■ [PREV] DRAW

This plot shows a *zero* of the expression—a value of X for which the value of the expression is zero. The zero is located where the graph of the expression crosses the horizontal axis.

Move the cross hairs to the approximate intersection of the graph and the horizontal axis. (Use [▲], [▼], [◀], and [▶] to move the cross hairs.)

Digitize this estimate for the zero.

[INS]
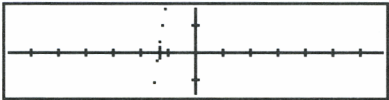
You'll use this point as an estimate for finding the exact zero of the expression. (In case the expression has more than one root, the estimate indicates which one you want.)
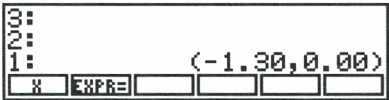
Return to the stack display.

[ON]

The coordinates of the digitized point, represented by a complex number, are in level 1. (Your coordinates may differ slightly from the illustration.)

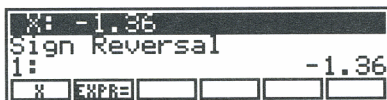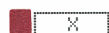Select the Solver menu.

[SOLV] SOLVR

The Solver menu shows all the variables in the current equation (only X in this example).
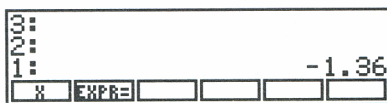
Store the digitized estimate in variable X.

```
┌─────────┐
│    X    │
└─────────┘
```

```
┌──────────────────────────────────────┐
│ X: (-1.30,0.00)              °         │
│2:                                      │
│1:                                      │
│ X  │EXPR=│   │   │   │   │             │
└──────────────────────────────────────┘
```

Although the digitized point contained two coordinates, the Solver will use only the first coordinate as an estimate.

Now solve for X.

```
■ ┌─────────┐
  │    X    │
  └─────────┘
```

```
┌──────────────────────────────────────┐
│ X: -1.36                               │
│Sign Reversal                           │
│1:                          -1.36       │
│ X  │EXPR=│   │   │   │   │             │
└──────────────────────────────────────┘
```

The message Sign Reversal indicates that the Solver found an approximate solution, correct to 12 digits. If the Solver found an exact solution, it would display the message Zero. These messages, called *qualifying messages*, are discussed in "SOLVE" in the Reference Manual.

Return to the normal stack display.

```
┌────┐
│ ON │
└────┘
```

```
┌──────────────────────────────────────┐
│3:                                      │
│2:                                      │
│1:                          -1.36       │
│ X  │EXPR=│   │   │   │   │             │
└──────────────────────────────────────┘
```

Finding a Minimum or Maximum

To find the zero of an expression, the Solver samples points on the graph, starting with your estimate, and tries to find points closer to the x-axis. If your estimate is quite close to a *positive local minimum* or a *negative local maximum*, there are no points nearby that are closer to the x-axis. In this case, the Solver finds that extremum (minimum or maximum) rather than a zero. (Generally the Solver won't "get stuck" at an extremum unless your estimate forces it there.)

Look at the graph you made in the last chapter, on page 96. It shows that the expression has a positive local minimum and a positive local maximum. The Solver can find the minimum, because locally it's the point *closest* to the x-axis; but the Solver can't find the maximum, because locally it's the point *farthest* from the x-axis.

In this section you'll plot the expression, using the plotting parameters stored in the variable PPAR1, then digitize three points to estimate the minimum, and then use the Solver to find a more accurate minimum.

Return the list stored in PPAR1 to the stack.

[USER] PPAR1

```
1: { (-1.50,1.20)
     (2.10,3.60) X 1.00
     (0.00,0.00) }
─────────────────────────
 X │FPAR│PPAR1│ EQ │PSUM│COT
```

Restore the variable PPAR to the values stored in PPAR1.

['] PPAR [STO]

```
3:
2:
1:                  -1.36
─────────────────────────
 X │FPAR│PPAR1│ EQ │PSUM│COT
```
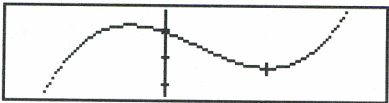
Plot the expression.

■[PLOT] DRAW

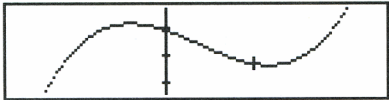Move the cross hairs to the approximate minimum.
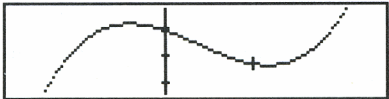

Digitize the point.
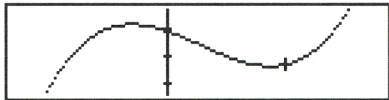
[INS]

Move the cross hairs just to the left of the minimum.

Digitize the point.

[INS]

Move the cross hairs just to the right of the minimum and digitize the point.

[INS]

Return to the stack display.

[ON]

```
3: (0.99,1.97)
2: (0.86,2.05)
1: (1.15,2.05)
STEQ RCEQ FMIN FMAX INDEP DRAW
```

The three points are in levels 1, 2, and 3. (Your points may differ slightly from the illustration.)

Now combine the three estimates in a list. By doing so, you can handle the three estimates as a single object. This is a typical use for lists—combining several objects into one.

■[LIST] 3 →LIST

```
1: { (0.99,1.97)
    (0.86,2.05)
    (1.15,2.05) }
→LIST LIST→ PUT GET PUTI GETI
```

Select the Solver menu.

[SOLV] SOLVR

```
1: { (0.99,1.97)
    (0.86,2.05)
    (1.15,2.05) }
  X EXPR=
```

The Solver menu shows all the variables in the current equation (only X in this example).

Store the list of points in the variable X.

```
┌─────────┐                    ┌──────────────────────────┐
│    X    │                    │X: ( (0.99,1.97) (0.8...   │
└─────────┘                    │2:                        │
                               │1:                  -1.36 │
                               │  X  EXPR=                 │
                               └──────────────────────────┘
```

The list of points is taken from the stack and stored in the variable X as initial estimates.

Solve for X.

```
┌──┬──────┐                    ┌──────────────────────────┐
│██│   X  │                    │X: 1.00                    │
└──┴──────┘                    │Extremum                   │
                               │1:                   1.00  │
                               │  X  EXPR=                  │
                               └──────────────────────────┘
```

The message E x t r e m u m indicates that the Solver found an extreme point of the expression.

Return to the normal stack display.

```
┌─────┐                        ┌──────────────────────────┐
│ ON  │                        │3:                         │
└─────┘                        │2:                  -1.36  │
                               │1:                   1.00  │
                               │  X  EXPR=                  │
                               └──────────────────────────┘
```

Calculate the extreme value.

```
┌─────────┐                    ┌──────────────────────────┐
│ EXPR=   │                    │EXPR=2.00                  │
└─────────┘                    │2:                   1.00  │
                               │1:                   2.00  │
                               │  X  EXPR=                  │
                               └──────────────────────────┘
```

The minimum value is 2.

Time Value of Money

This section shows how to use the Solver with time value of money (TVM) calculations. For n number of periods, $i\%$ interest per period, pmt payment, pv present value, and fv future value, the formula for TVM is:

$$(1 - sppv) \times pmt \times (100/i) + pv = -fv \times sppv$$

where

$$sppv \; (single \; payment \; present \; value) = (1 + i/100)^{-n}$$

$$= \exp{(-n \times \ln{(1 + i/100)})}.$$

This formula assumes that payments are made at the end of each period.

Here are the major steps you'll perform:

1. Key in the expression for *sppv* and store it in a variable SPPV.

2. Key in the equation and store it in a variable TVM.

3. Make TVM the current equation.

4. Use the Solver to calculate any of the five variables *n*, *i*, *pmt*, *pv*, or *fv*, for given values of the other four variables.

Before starting, clear the stack and select FIX 2 number display mode.

■ CLEAR
■ MODE 2 FIX

```
3:
2:
1:
STO  FIX■  SCI  ENG  DEG  RAD■
```

Step 1: Key in the expression for *sppv* and store it in a variable SPPV.

Key in the expression for *sppv*.

' ■ LOGS EXP − N × LNP1
I ÷ 100 ENTER

```
2:
1: 'EXP(-N*LNP1(I/100))
LOG  ALOG  LN  EXP  LNP1  EXPM
```

This expression takes advantage of the greater accuracy of LNP1 to calculate $\ln{(1 + i/100)}$.

Create the variable SPPV and check the USER menu.

' SPPV STO
USER

```
3:
2:
1:
SPPV  X  PPAR  PPAR1  EQ  PSUM
```

Step 2: Key in the equation and store it in a variable TVM.

Key in the equation for TVM.

`'` `(` `1` `−` `SPPV` `▮` `)` `×` PMT `×`
100 `÷` I `+` PV `=` `−` FV `×` `SPPV`
`ENTER`

```
2:
1: '(1-SPPV)*PMT*100/I+
      PV=-FV*SPPV'
SPPV    X    PPAR  PPAR1   EQ   PSUM
```

Create the variable TVM.

`'` TVM `STO`

```
3:
2:
1:
TVM   SPPV    X    PPAR  PPAR1   EQ
```

The USER menu shows a new label for TVM.

Step 3: Make TVM the current equation.

Key in the name TVM.

`'` `TVM`

```
2:
1:
'TVM▯
TVM   SPPV    X    PPAR  PPAR1   EQ
```

Store the name TVM in the variable EQ.

`SOLV` `STEQ`

```
3:
2:
1:
STEQ  RCEQ  SOLVR  ISOL  QUAD  SHOW
```

Step 4: Use the Solver to calculate any of the five variables *n*, *i*, *pmt*, *pv*, or *fv*, for given values of the other four variables.

Select the Solver menu.

`SOLVR`

```
3:
2:
1:
 N     I    PMT    PV    FV   LEFT=
```

All the variables in TVM and SPPV appear in the menu. (The variables in SPPV appear because the current equation, TVM, contains SPPV.)

Given values N = 30 × 12, I = 11.5/12, PMT = −630, and FV = 0, calculate PV. (PMT has a negative value because money paid out is a negative number, while money received is a positive number.)

First assign the value to N.

30 [ENTER] 12 [×] [N]

```
N: 360.00
2:
1:
N    I   PMT   PV   FV  LEFT=
```

Assign the value to I.

11.5 [ENTER] 12 [÷] [I]

```
I: 0.96
2:
1:
N    I   PMT   PV   FV  LEFT=
```

Assign the value to PMT.

−630 [PMT]

```
PMT: -630.00
2:
1:
N    I   PMT   PV   FV  LEFT=
```

Assign the value to FV.

0 [FV]

```
FV: 0.00
2:
1:
N    I   PMT   PV   FV  LEFT=
```

Now solve for PV.

■ [PV]

```
PV: 63617.64
Zero
1:              63617.64
N    I   PMT   PV   FV  LEFT=
```

The message Zero indicates that the returned value exactly satisfies the current equation.

9

Symbolic Solutions

This chapter describes two methods for finding symbolic solutions. There is a simple method for solving a quadratic expression by calculating the linear expression that represents both zeros. There is also a more versatile method that provides a symbolic solution for a variable in more general equations.

Each method works with both expressions and equations. The *zero* of an expression $f(x)$ is the same as the *root* of the equation $f(x) = 0$, and the *root* of the equation $f(x) = g(x)$ is the same as the *zero* of the expression $f(x) - g(x)$.

Finding the Zeros of a Quadratic Expression

You can find both zeros of a quadratic expression without plotting or making estimates. The following example solves $x^2 - 6x + 8$.

Before starting the example, clear the stack and select STD number display mode.

■ [CLEAR]
■ [MODE] STD

```
3:
2:
1:
STD■ | FIX  | SCI  | ENG  | DEG  | RAD■
```

Put the expression on the stack.

['] X ■ [^] 2 [−] 6 [×] X [+] 8 [ENTER]

```
3:
2:
1:              'X^2-6*X+8'
STD■ | FIX  | SCI  | ENG  | DEG  | RAD■
```
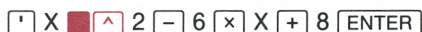
Put the name X on the stack, indicating the variable for which you're solving.

`'` X `ENTER`

```
3:
2:                    'X^2-6*X+8'
1:                           'X'
STD■  FIX   SCI   ENG   DEG  RAD■
```

Calculate the zeros, using QUAD (*quadratic*) in the SOLVE menu.

`SOLV` QUAD

```
3:
2:
1:                     '(6+s1*2)/2'
STEQ RCEQ SOLVR ISOL QUAD SHOW
```

This expression represents both solutions to the quadratic expression. The variable s1 represents an arbitrary sign, either $+1$ or -1, and each value of s1 corresponds to a zero of the expression.

Store the expression as the current equation.

STEQ

```
3:
2:
1:
STEQ RCEQ SOLVR ISOL QUAD SHOW
```

Display the Solver menu.

SOLVR

```
3:
2:
1:
S1  EXPR=
```

s1 is the only variable in the current equation.

First make s1 a positive sign.

1 `S1`

```
s1: 1
2:
1:
S1  EXPR=
```

Return one of the solutions to level 1.

`EXPR=`

```
EXPR=4
2:
1:                                 4
S1  EXPR=
```

Now make s1 a negative sign.

−1 `s1`

```
s1: -1
2:
1:                    4
 s1  EXPR=
```

Return the second solution to level 1.

`EXPR=`

```
EXPR=2
2:                    4
1:                    2
 s1  EXPR=
```

The two roots of $x^2 - 6x + 8$ are $x = 4$ and $x = 2$.

Isolating a Variable

The HP-28S can isolate a single occurrence of a variable in an equation, returning an expression representing the symbolic solution of the equation. In other words, if x is the variable for which the equation is solved, and a, b, and c are the other variables in the equation, isolating x produces an expression in a, b, and c such that the equation is satisfied when x has the value of the expression.

For the first example, isolate x in the equation

$$a(x + 3) - b = c.$$

This example is simple because there is only one occurrence of x. Later examples show how to manipulate the equation to produce a single occurrence of x.

Clear the stack.

■ `CLEAR`

```
3:
2:
1:
 s1  EXPR=
```

Put the equation on the stack.

`'` A `×` `(` X `+` 3 ■`)` `−` B `=` C
`ENTER`

```
3:
2:
1:         'A*(X+3)-B=C'
 s1  EXPR=
```

Specify the variable you want to isolate.

`'` X

```
2:
1:                              'A*(X+3)-B=C'
'X▉
  S1   EXPR=
```

Isolate x, using ISOL (*isolate*) in the SOLVE menu.

SOLV ISOL

```
3:
2:
1:                                  '(C+B)/A-3'
STEQ  RCEQ  SOLVR  ISOL  QUAD  SHOW
```

The expression returned represents a symbolic solution of the equation for x—that is, the equation

$$a(x + 3) - b = c$$

is satisfied when $x = (c + b)/a - 3$.

Expanding and Collecting

If x occurs more than once, you must manipulate the equation to eliminate all but one occurrence of x. The next example shows how to isolate x in the equation

$$2(a + x) = 3(b - x) + c.$$

The strategy for this example is to expand the equation, subtract one side's x-term from both sides, collect the equation to cancel the x-term on one side and produce a single x-term on the other side, and then isolate.

Put the equation on the stack.

`'` 2 ⊠ (| A ⊞ X ▉|) ⊟ 3 ⊠ (| B
⊟ X ▉|) ⊞ C ENTER

```
3:
2:                                  '(C+B)/A-3'
1:             '2*(A+X)=3*(B-X)+C'
STEQ  RCEQ  SOLVR  ISOL  QUAD  SHOW
```

Select the ALGEBRA menu.

▉ ALGBRA

```
3:
2:                                  '(C+B)/A-3'
1:             '2*(A+X)=3*(B-X)+C'
COLCT  EXPAN  SIZE  FORM  OBSUB  EXSUB
```

In this example you'll use EXPAN (*expand*) and COLCT (*collect*) to manipulate the equation. In the next example you'll use FORM (*form algebraic expression*) to manipulate an equation. All commands in the ALGEBRA menu are described briefly in appendix D, "Menu Map." For complete descriptions, refer to "ALGEBRA" in the Reference Manual. In addition, FORM, a powerful algebraic editor, has its own section "ALGEBRA (FORM)" in the Reference Manual.

Expand both sides of the equation.

EXPAN

```
3:
2:                  '(C+B)/A-3'
1:      '2*A+2*X=3*B-3*X+C'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

To subtract the left side's x-term ($2x$) from both sides of the equation, first put the left side's x-term on the stack.

' 2 × X ENTER

```
3:                  '(C+B)/A-3'
2:      '2*A+2*X=3*B-3*X+C'
1:                        '2*X'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Then subtract $2x$ from both sides.

–

```
2:                  '(C+B)/A-3'
1:  '2*A+2*X-2*X=3*B-3*X
  +C-2*X'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Collect the equation.

COLCT

```
3:
2:                  '(C+B)/A-3'
1:          '2*A=3*B+C-5*X'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Each side is collected independently, and the x-terms cancel on the left side.

Now you can isolate x in the equation. Specify the variable you want to isolate.

' X

```
2:                  '(C+B)/A-3'
1:          '2*A=3*B+C-5*X'
'X▯
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Isolate x. The command ISOL appears in the second row of the ALGEBRA menu as well as the SOLVE menu.

NEXT ISOL

```
3:
2:                        '(C+B)/A-3'
1:                 '(3*B+C-2*A)/5'
TAYLR ISOL QUAD SHOW OBGET EXGET
```

The expression returned represents a symbolic solution of the equation for x—that is, the equation

$$2(a + x) = 3(b - x) + c$$

is satisfied when $x = (3b + c - 2a)/5$.

Using FORM

If there are multiple occurrences of x, and if any occurrence has a symbolic coefficient, the command COLCT won't combine the coefficients. The next example isolates x in the equation

$$a(x + b) + 2x = c,$$

where x occurs more than once and has a symbolic coefficient a. The strategy is to expand the equation, use FORM to collect coefficients of x, and then isolate x.

Put the equation on the stack.

' A × (X + B ■) + 2 × X =
C ENTER

```
3:                        '(C+B)/A-3'
2:                 '(3*B+C-2*A)/5'
1:                 'A*(X+B)+2*X=C'
TAYLR ISOL QUAD SHOW OBGET EXGET
```

Expand the equation.

NEXT EXPAN

```
3:                        '(C+B)/A-3'
2:                 '(3*B+C-2*A)/5'
1:                 'A*X+A*B+2*X=C'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Now use FORM to collect the coefficients of x.

FORM

```
((((■*X)+(A*B))+(2*X))=
C)
COLCT EXPAN LEVEL EXGET [←] [→]
```

Normal calculator operation is suspended while FORM is active. The FORM display shows the equation with all subexpressions delimited by parentheses. You'll use FORM to manipulate subexpressions within the equation.

The goal is to combine $(A*X)$ and $(2*X)$ in a single term $((A+2)*X)$. There are three steps required, shown below as you might write them on paper. The current form of the equation is:

$$(ax + ab) + 2x = c$$

The first step is to commute ax and ab, giving:

$$(ab + ax) + 2x = c$$

The second step is to associate ax and $2x$, giving:

$$ab + (ax + 2x) = c$$

The third step is to merge ax and $2x$, giving:

$$ab + (a + 2) x = c$$

Step 1: Commute ax and ab.

Move the cursor (the inverse character or characters) to +.

[→]   [→]   [→]

```
(((A*X)█(A*B))+(2*X))=
C)
COLCT EXPAN LEVEL EXGET [←]  [→]
```

The position of the cursor determines which subexpression you're acting on. Here you want to act on the subexpression $((A*X)+(A*B))$ to commute the arguments to +.

Display the first row of manipulations for +.

NEXT

```
(((A*X)⊞(A*B))+(2*X))=
C)
 -() | ↔ | ←M | M→ | ←A | A→
```

The manipulations that appear when you press NEXT are specific to the function or variable indicated by the cursor; these manipulations are specific to +.

Commute the arguments to +, using ↔ (*commute*).

↔

```
(((A*B)⊞(A*X))+(2*X))=
C)
 -() | ↔ | ←M | M→ | ←A | A→
```

Return to the main FORM menu.

ENTER

```
(((A*B)⊞(A*X))+(2*X))=
C)
COLCT|EXPAN|LEVEL|ENGET| [←] | [→]
```

Step 2: Associate *ax* and 2*x*.

Move the cursor to the second +.

[→] [→] [→] [→]

```
(((A*B)+(A*X))⊞(2*X))=
C)
COLCT|EXPAN|LEVEL|ENGET| [←] | [→]
```

Here you want to act on the subexpression

$$(((A*B)+(A*X))+(2*X))$$

to associate the terms (A*X) and (2*X) in a single subexpression. Display the first row of manipulations for +.

NEXT

```
(((A*B)+(A*X))⊞(2*X))=
C)
 -() | ↔ | ←M | M→ | ←A | A→
```

These are the same manipulations as before because the cursor again indicated an additive subexpression.

Associate the terms ⟨A*X⟩ and ⟨2*X⟩ in the subexpression ⟨⟨A*X⟩+⟨2*X⟩⟩, using ▓A→▓ (*associate right*).

A→

```
{{(A*B)█((A*X)+(2*X)))=
C}
  -() | ←→ | ←M | M→ | ←A | A→
```

Return to the main FORM menu.

[ENTER]

```
{{(A*B)█((A*X)+(2*X)))=
C}
COLCT EXPAN LEVEL EXGET [←] [→]
```

Step 3: Merge *ax* and 2*x*.

Move the cursor to the second +.

[→] [→] [→] [→]

```
{{(A*B)+((A*X)█(2*X)))=
C}
COLCT EXPAN LEVEL EXGET [←] [→]
```

Here you want to act on the subexpression ⟨⟨A*X⟩+⟨2*X⟩⟩ to combine the coefficients of X.

Display the first row of manipulations for +.

[NEXT]

```
{{(A*B)+((A*X)█(2*X)))=
C}
  -() | ←→ | ←M | M→ | ←A | A→
```

Combine the coefficients of X, using ▓M→▓ (*merge right*).

M→

```
{{(A*B)+((A+2)█X))=C}
 1/() | ←→ | ←D | D→ | ←A | A→
```

This accomplishes the goal of combining ⟨A*X⟩ and ⟨2*X⟩ in a single term ⟨⟨A+2⟩*X⟩.

Exit FORM and return the modified equation to the stack.

[ON]

```
3:          '(C+B)/A-3'
2:        '(3*B+C-2*A)/5'
1:       'A*B+(A+2)*X=C'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Now that x occurs only once in the equation, you can isolate x. Specify the variable you want to isolate.

⌐'⌐ X

```
2:            '(3*B+C-2*A)/5'
1:            'A*B+(A+2)*X=C'
'X▊
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Isolate x.

NEXT   ISOL

```
3:                '(C+B)/A-3'
2:            '(3*B+C-2*A)/5'
1:            '(C-A*B)/(A+2)'
TAYLR ISOL QUAD SHOW OBGET EXGET
```

The expression returned represents a symbolic solution of the equation for x—that is, the equation

$$a(x + b) + 2x = c$$

is satisfied when $x = (c - ab)/(a + 2)$.

10

Calculus

You can symbolically differentiate any expression for which a sensible derivative exists. Integration is more restricted: you can compute a *definite numerical integral* for any expression, but an *exact symbolic integral* only for a polynomial.

This chapter contains simple examples of finding derivatives, indefinite integrals, and definite integrals for expressions. For more information about doing calculus on the HP-28S, refer to "Calculus" in the Reference Manual.

Differentiating an Expression

You can differentiate an expression step-by-step, observing how the calculator applies the rules of differentiation, or you can differentiate an expression all at once. The final results are identical. In this section you'll differentiate an expression twice, first step-by-step and then all at once.

Step-by-Step Differentiation

To differentiate step-by-step, key in the derivative as a expression. For this example, calculate:

$$\frac{d}{dx} \tan (x^2 + 1)$$

Before starting the example, clear the stack, select Radians angle mode, and select STD number display mode.

■ [CLEAR]
■ [MODE] RAD
  STD

```
3:
2:
1:
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Purge the variable X (if it exists).

['] X ■ [PURGE]

```
3:
2:
1:
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Now start the expression for the derivative, beginning with the variable of differentiation.

['] ■ [d/dx] X [(]

```
2:
1:
'∂X(⊟
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Next, key in the expression to be differentiated.

[TRIG] TAN X ■ [^] 2 [+] 1 [ENTER]

```
3:
2:
1:          '∂X(TAN(X^2+1))'
SIN | ASIN | COS | ACOS | TAN | ATAN
```

This expression represents the derivative, with respect to x, of $\tan (x^2 + 1)$.

Evaluate the expression once.

[EVAL]

```
2:
1: '(1+SQ(TAN(X^2+1)))*
   ∂X(X^2+1)'
SIN | ASIN | COS | ACOS | TAN | ATAN
```

The result reflects the chain rule of differentiation:

$$\frac{d}{dx} \tan (x^2 + 1) = \frac{d}{d(x^2 + 1)} \tan (x^2 + 1) \times \frac{d}{dx} (x^2 + 1)$$

The derivative of the tangent function has been evaluated. Next you'll evaluate the derivative of $x^2 + 1$.

Evaluate the expression a second time.

EVAL

```
2:
1: '(1+SQ(TAN(X^2+1)))*
    ∂X(X^2)'
 SIN  ASIN  COS  ACOS  TAN  ATAN
```

The result reflects the derivative of a sum:

$$\frac{d}{dx} (x^2 + 1) = \frac{d}{dx} x^2 + \frac{d}{dx} 1$$

The derivative of 1 is 0, so that term disappears. Next you'll evaluate the derivative of x^2.

Evaluate the expression a third time.

EVAL

```
2:
1: '(1+SQ(TAN(X^2+1)))*
    (∂X(X)*2*X^(2-1))'
 SIN  ASIN  COS  ACOS  TAN  ATAN
```

The result again 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. Finally, evaluate the derivative of x itself.

Evaluate the expression a fourth time.

EVAL

```
2:
1: '(1+SQ(TAN(X^2+1)))*
    (2*X)'
 SIN  ASIN  COS  ACOS  TAN  ATAN
```

Here is the fully evaluated derivative.

Complete Differentiation

To differentiate an expression all at once, perform differentiation as a stack operation. Again, suppose you want to find:

$$\frac{d}{dx} \tan (x^2 + 1)$$

Put the expression to be differentiated on the stack.

■ CLEAR
' TAN X ■ ^ 2 + 1 ENTER

```
3:
2:
1:              'TAN(X^2+1)'
 SIN  ASIN  COS  ACOS  TAN  ATAN
```

Specify the variable of differentiation.

' X ENTER

```
3:
2:              'TAN(X^2+1)'
1:                       'X'
 SIN  ASIN  COS  ACOS  TAN  ATAN
```

Differentiate the expression.

■ d/dx

```
2:
1:  '(1+SQ(TAN(X^2+1)))*
    (2*X)'
 SIN  ASIN  COS  ACOS  TAN  ATAN
```

The fully evaluated derivative is returned to level 1.

Integrating an Expression

The HP-28S calculates the indefinite integral of an expression by *symbolic integration*, which returns an expression as a result. This method returns an exact result only for polynomial expressions. (For other expressions, the HP-28S integrates a Taylor series approximation to the expression. See "Calculus" in the Reference Manual for details.) The first example below demonstrates symbolic integration.

In contrast, definite integrals are calculated by *numerical integration*, which returns numerical results. This method works for any expression that is "well-behaved" in the mathematical sense. The second example below demonstrates numerical integration.

Symbolic Integration of Polynomials

In this example you'll symbolically integrate the polynomial

$$8x^3 + 9x^2 + 2x + 5.$$

Clear the stack.

■ CLEAR

```
3:
2:
1:
SIN  ASIN  COS  ACOS  TAN  ATAN
```

Put the polynomial on the stack.

' 8 × X ■ ^ 3 + 9 × X ■ ^ 2
+ 2 × X + 5 ENTER

```
3:
2:
1:     '8*X^3+9*X^2+2*X+5'
SIN  ASIN  COS  ACOS  TAN  ATAN
```

Specify the variable of integration.

' X ENTER

```
3:
2:     '8*X^3+9*X^2+2*X+5'
1:                     'X'
SIN  ASIN  COS  ACOS  TAN  ATAN
```

Specify the degree of the polynomial.

3 ENTER

```
3:     '8*X^3+9*X^2+2*X+5'
2:                     'X'
1:                      3
SIN  ASIN  COS  ACOS  TAN  ATAN
```

Integrate the polynomial.

■ ∫

```
2:
1:  '5*X+X^2+3*X^3+2*X^4
SIN  ASIN  COS  ACOS  TAN  ATAN
```

Wait for the ((●)) annunciator to disappear, indicating that integration is completed. The integral is returned to level 1.

Numerical Integration of Expressions

In this example you'll find a numerical value for the integral

$$\int_0^1 \exp(x^3 + 2x^2 - x + 4)dx$$

Clear the stack.

■ CLEAR

```
3:
2:
1:
 SIN  ASIN  COS  ACOS  TAN  ATAN
```

Put the expression on the stack.

' ■LOGS EXP X ■^ 3 ⊞ 2
⊠ X ■^ 2 ⊟ X ⊞ 4 ENTER

```
3:
2:
1:  'EXP(X^3+2*X^2-X+4)'
 LOG  ALOG  LN   EXP  LNP1 EXPM
```

Key in the variable and limits of integration. You'll enter them as ob-
jects within a list object. (This is a typical use of a list—combining
several objects so you can handle them as a single object.)

⦅ X SPACE 0 SPACE 1 ENTER

```
3:
2:  'EXP(X^3+2*X^2-X+4)'
1:          { X 0 1 }
 LOG  ALOG  LN   EXP  LNP1 EXPM
```

X is the variable of integration, 0 and 1 the limits of integration.

Next key in the accuracy you require.

If the expression included constants derived from empirical data,
specify the accuracy of the constants. For example, if the constants are
accurate to three decimal places, specify an accuracy of .001.

In this example you're integrating an expression without empirical constants, so you could specify 12-digit accuracy. However, the iterative process of numerical integration takes longer for greater accuracy, so here you'll specify an accuracy of .00001.

1E−5 [ENTER]

```
3: 'EXP(X^3+2*X^2-X+4)'
2:              { X 0 1 }
1:                 .00001
 LOG | ALOG |  LN  |  EXP |  LNP1 | EXPM
```

Find the integral.

■ ∫

```
3:
2:            103.117678153
1:        1.03086911923E-3
 LOG | ALOG |  LN  |  EXP |  LNP1 | EXPM
```

The estimated integral is returned to level 2, and an error bound is returned to level 1.

The value of the integral is 103.118 ± .001. Note that the error bound returned is approximately the product of the estimated integral and the accuracy you specified.

11

Vectors and Matrices

The HP-28S deals with two types of arrays: *vectors*, which are one-dimensional arrays, and *matrices*, which are two-dimensional arrays. You can enter vectors and matrices as individual objects, called *array objects*, and calculate with them as easily as with numbers.

This chapter shows basic array calculations using real arrays—vectors and matrices whose elements are real numbers. You can also calculate with arrays whose elements are complex numbers.

All commands in the ARRAY menu are described briefly in appendix D, "Menu Map." For complete descriptions, refer to "ARRAY" in the Reference Manual.

Vectors

This section demonstrates vector arithmetic, the cross product, and the dot product.

Keying In a Vector.

Before beginning these examples, clear the stack and select STD number display mode.

■ CLEAR
■ MODE STD

```
3:
2:
1:
STD■ | FIX | SCI | ENG | DEG | RAD■
```
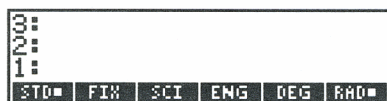
Key in the vector [2 3 4]. You can use either spaces or the non-radix mark (the comma if you have selected the period as the radix mark) to separate 2 from 3 and 3 from 4.

[ ] 2,3,4 [ENTER]

```
3:
2:
1:              [ 2 3 4 ]
STO■ FIX  SCI  ENG  DEG  RAD■
```

Multiplying and Dividing a Vector by a Number

Multiply the vector by 15.

15 [×].

```
3:
2:
1:           [ 30 45 60 ]
STO■ FIX  SCI  ENG  DEG  RAD■
```

For multiplication, the order of the arguments makes no difference, just as it makes no difference when you multiply two numbers. However, for division, the vector must be in level 2 and the number in level 1.

Divide the vector by 5.

5 [÷]

```
3:
2:
1:            [ 6 9 12 ]
STO■ FIX  SCI  ENG  DEG  RAD■
```

Adding and Subtracting Vectors

You can add and subtract vectors just as you add and subtract numbers, provided that the vectors have the same number of elements. For subtraction, the order of the arguments is important, just as it's important when you subtract one number from another.

For this example, subtract the vector [−10 20 30].

[ ] −10,20,30 [−]

```
3:
2:
1:        [ 16 −11 −18 ]
STO■ FIX  SCI  ENG  DEG  RAD■
```

Finding the Cross Product

Find the cross product of the vector in level 1 with the vector [2 −2 1]. (The cross product is defined only for two- and three-element vectors.)

Key in the vector.

⊞ 2,−2,1

```
2:
1:        [ 16 -11 -18 ]
[2,-2,1□
STD■  FIX  SCI  ENG  DEG  RAD■
```

Calculate the cross product, using CROSS in the third row of the ARRAY menu.

■ ARRAY NEXT NEXT CROSS

```
3:
2:
1:        [ -47 -52 -10 ]
CROSS  DOT  DET  ABS  RNRM CNRM
```

Finding the Dot Product

Find the dot product of the vector in level 1 with the vector [5 7 2]. (The two vectors must have the same number of elements.)

Key in the vector.

⊞ 5,7,2

```
2:
1:        [ -47 -52 -10 ]
[5,7,2□
CROSS  DOT  DET  ABS  RNRM CNRM
```

Calculate the dot product.

DOT

```
3:
2:
1:                  -619
CROSS  DOT  DET  ABS  RNRM CNRM
```

Matrices

This section describes how to invert a matrix and how to find the determinant of a matrix. Both of these calculations are restricted to *square* matrices—those with the same number of rows as columns.

The calculations you performed on vectors also apply to matrices (with the exception of the dot and cross products). You can multiply or divide a matrix by a number, and you can add or subtract two matrices (provided that the matrices have the same dimensions).

Keying In a Matrix

Key in the following matrix:

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

Start the matrix.

[]

```
2:
1:                    -619
[□
CROSS  DOT   DET   ABS  RNRM CNRM
```

Enter each row of the matrix like a separate vector.

[] 1,2,3
[] 1,3,3
[] 1,2,4 [ENTER]

```
1:  [[ 1 2 3 ]
     [ 1 3 3 ]
     [ 1 2 4 ]]
CROSS  DOT   DET   ABS  RNRM CNRM
```

Viewing a Large Matrix

When a matrix has many elements or non-integer elements, you may not see the entire matrix at once. To view a large matrix, use ■[EDIT] (if the matrix is in level 1) or ■[VISIT] to return the matrix to the command line. You can then use the cursor menu keys to display any part of the matrix. For details, refer to "Editing Existing Objects" in chapter 18.

Inverting a Matrix

Because the matrix in level 1 is square, you can find its inverse.

■ ▭1/x▭

```
1: [[ 6 -2 -3 ]
    [ -1  1  0 ]
    [ -1  0  1 ]]
CROSS  DOT  DET  ABS  RNRM  CNRM
```

Finding the Determinant

Because the matrix in level 1 is square, you can find its determinant.

▭ DET ▭

```
3:
2:              -619
1:                 1
CROSS  DOT  DET  ABS  RNRM  CNRM
```

Multiplying Two Arrays

You can use the ▭×▭ function to multiply two matrices or a matrix and a vector. (Use CROSS or DOT to multiply two vectors, as described above.)

Multiplying Two Matrices

The order of the arguments is important when multiplying two matrices. The number of *columns* in the matrix in level 2 must equal the number of *rows* in the matrix in level 1. For example, you can calculate the following matrix product.

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

To calculate this matrix product:

Enter the first matrix.

⑨ ⑨ 2,2
⑨ 4,1
⑨ 2,3 ⌈ENTER⌉

```
1: [[ 2  2 ]
    [ 4  1 ]
    [ 2  3 ]]
CROSS  DOT  DET  ABS  RNRM  CNRM
```

Key in the second matrix.

⑨ ⑨ 2,2,1,4
⑨ 3,4,2,1

```
1: [[ 2  2 ]
    [ 4  1 ]
[[2,2,1,4[3,4,2,1□
CROSS  DOT  DET  ABS  RNRM  CNRM
```

Multiply the matrices.

⌈×⌉

```
1: [[ 10  12  6  10 ]
    [ 11  12  6  17 ]
    [ 13  16  8  11 ]]
CROSS  DOT  DET  ABS  RNRM  CNRM
```

Multiplying a Matrix and a Vector

The order of the arguments is important when multiplying a matrix and a vector. The matrix must be in level 2, and the vector must be in level 1. The number of *columns* in the matrix must equal the number of *elements* in the vector.

For the next example, multiply the matrix currently in level 1 by the vector [3 1 1 2].

Key in the vector.

⑨ 3,1,1,2

```
1: [[ 10  12  6  10 ]
    [ 11  12  6  17 ]
[3,1,1,2□
CROSS  DOT  DET  ABS  RNRM  CNRM
```

Multiply the matrix and vector.

⌈×⌉

```
3:                    -619
2:                       1
1:           [ 68  85  85 ]
CROSS  DOT  DET  ABS  RNRM  CNRM
```

Solving a System of Linear Equations

To solve a system of n linear equations with n variables, use an n-element *constant* vector, an $n \times n$ *coefficient* matrix, and division ($\boxed{\div}$). The constant vector contains the constant values of the equations. The coefficient matrix contains the coefficients of the variables.

The next example shows how to solve a system of three linearly independent equations in three variables. Suppose the equations are

$$3x + y + 2z = 13$$
$$x + y - 8z = -1$$
$$-x + 2y + 5z = 13$$

Enter the constant vector.

$\boxed{[}$ 13, −1,13 $\boxed{\text{ENTER}}$

```
3:                              1
2:                  [ 68 85 85 ]
1:                  [ 13 -1 13 ]
CROSS  DOT   DET   ABS  RNRM CNRM
```

Key in the coefficient matrix.

$\boxed{[}$ $\boxed{[}$ 3,1,2
$\boxed{[}$ 1,1,−8
$\boxed{[}$ −1,2,5

```
2:                  [ 68 85 85 ]
1:                  [ 13 -1 13 ]
[[3,1,2[1,1,-8[-1,2,5▯
CROSS  DOT   DET   ABS  RNRM CNRM
```

Solve the system of equations.

$\boxed{\div}$

```
3:                              1
2:                  [ 68 85 85 ]
1:                     [ 2 5 1 ]
CROSS  DOT   DET   ABS  RNRM CNRM
```

The values in the solution vector are the values of the variables that satisfy the equations:

$$x = 2, \quad y = 5, \quad z = 1$$

To solve under-determined, over-determined, or near-singular systems of equations, refer to "ARRAY" in the Reference Manual.

12

Statistics

This chapter describes how to enter statistical data and how to calculate single-sample and paired-sample statistics, using commands in the STAT menu. All commands in the STAT menu are described briefly in appendix D, "Menu Map." For complete descriptions, refer to "STAT" in the Reference Manual.

The following table lists the consumer price index change (CPI), the producer price index change (PPI), and the unemployment rate (UR), all in percentages, for the United States over a 5-year period. Enter these data and calculate statistics from them.

Data for Statistical Example

Year	CPI	PPI	UR
1975	9.1	9.2	8.5
1976	5.8	4.6	7.7
1977	6.5	6.1	7.0
1978	7.6	7.8	6.0
1979	11.5	19.3	5.8

Entering Data

Statistical data are stored in a statistics matrix named ΣDAT—an ordinary matrix with a special name. Each row of the matrix contains one data point, which in this example comprises the values of CPI, PPI, and UR for one year.

Before you start, clear the stack and select FIX 2 number display mode.

■ CLEAR
■ MODE 2 FIX

```
3:
2:
1:
STD │ FIX■ │ SCI │ ENG │ DEG │ RAD■
```

Clear any previous statistical data, using CLΣ (*clear statistics*) in the STAT menu. (Any existing ΣDAT is purged.)

■ STAT CLΣ

```
3:
2:
1:
Σ+ │ Σ- │ NΣ │ CLΣ │ STOΣ │ RCLΣ
```

Key in the data point for 1975.

⬚ 9.1,9.2,8.5

```
2:
1:
[9.1,9.2,8.5□
Σ+ │ Σ- │ NΣ │ CLΣ │ STOΣ │ RCLΣ
```
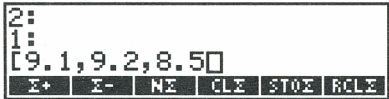
Store this data point in ΣDAT.

   Σ+

```
3:
2:
1:
Σ+ │ Σ- │ NΣ │ CLΣ │ STOΣ │ RCLΣ
```
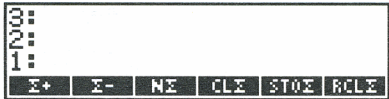
A new matrix named ΣDAT is automatically created. The data point for 1975 is the first row of ΣDAT.

Enter the data point for 1976.

⬚ 5.8,4.6,7.7 Σ+

```
3:
2:
1:
Σ+ │ Σ- │ NΣ │ CLΣ │ STOΣ │ RCLΣ
```
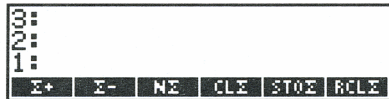
The data point for 1976 is added to ΣDAT, forming the second row of the statistics matrix.

Enter the data point for 1977.

⊞ 6.5,6.1,7 Σ+

```
3:
2:
1:
‾Σ+‾|‾Σ-‾|‾NΣ‾|‾CLΣ‾|‾STOΣ‾|‾RCLΣ‾
```

The data point for 1977 is added to ΣDAT, forming the third row of the statistics matrix.

Editing Data

If you make a mistake while keying in data, and you realize your mistake before pressing Σ+ , you can simply edit the command line. But suppose you believe that you made a mistake entering the data point for 1976. You can return data points to the stack, edit those that contain mistakes, and restore the data points to ΣDAT.

Remove the data point for 1977 (the last row in ΣDAT) and return it to the stack.

Σ-

```
3:
2:
1:     [ 6.50 6.10 7.00 ]
‾Σ+‾|‾Σ-‾|‾NΣ‾|‾CLΣ‾|‾STOΣ‾|‾RCLΣ‾
```

Remove the data point for 1976 (the last row in ΣDAT) and return it to the stack.

Σ-

```
3:
2:     [ 6.50 6.10 7.00 ]
1:     [ 5.80 4.60 7.70 ]
‾Σ+‾|‾Σ-‾|‾NΣ‾|‾CLΣ‾|‾STOΣ‾|‾RCLΣ‾
```

If you find you did make a mistake in this data point, press ■[EDIT] to return the data point to the command line, edit the data point, and press [ENTER] to put the corrected data point back on the stack. (Refer to "Editing Existing Objects" in chapter 18.)

Return the corrected data point for 1976 to ΣDAT.

Σ+ .

```
3:
2:
1:     [ 6.50 6.10 7.00 ]
‾Σ+‾|‾Σ-‾|‾NΣ‾|‾CLΣ‾|‾STOΣ‾|‾RCLΣ‾
```

Return the data point for 1977 to ΣDAT.

Σ+ .

```
3:
2:
1:
[ Σ+ | Σ- | NΣ | CLΣ |STOΣ|RCLΣ]
```

Now enter the rest of the data (for 1978 and 1979) and check that you entered all five data points.

[ ] 7.6,7.8,6   Σ+
[ ] 11.5,19.3,5.8   Σ+
NΣ

```
3:
2:
1:                        5.00
[ Σ+ | Σ- | NΣ | CLΣ |STOΣ|RCLΣ]
```

Single-Sample Statistics

In this section you'll find the mean, standard deviation, and variance of CPI, PPI, and UR. The data for CPI are contained in the first column of ΣDAT, the data for PPI in the second column, and the data for UR in the third column.

Display the second row of the STAT menu.

[NEXT]

```
3:
2:
1:                        5.00
[ TOT |MEAN|SDEV| VAR |MAXΣ|MINΣ]
```

Here are the commands for mean, standard deviation, and variance.

Finding the Mean

Calculate the mean.

MEAN

```
3:                        5.00
2:
1:     [ 8.10 9.40 7.00 ]
[ TOT |MEAN|SDEV| VAR |MAXΣ|MINΣ]
```

The mean for CPI is 8.1, for PPI is 9.4, and for UR is 7.

Finding the Standard Deviation

Calculate the standard deviation.

SDEV

```
3:                    5.00
2:    [ 8.10 9.40 7.00 ]
1:    [ 2.27 5.80 1.14 ]
  TOT  MEAN SDEV  VAR  MAXΣ MINΣ
```

The sample standard deviation for CPI is 2.27, for PPI is 5.8, and for UR is 1.14.

Finding the Variance

Calculate the variance.

VAR

```
3:    [ 8.10 9.40 7.00 ]
2:    [ 2.27 5.80 1.14 ]
1:    [ 5.17 33.64 1.30 ]
  TOT  MEAN SDEV  VAR  MAXΣ MINΣ
```

The sample variance for CPI is 5.17, for PPI is 33.64, and for UR is 1.3.

Paired-Sample Statistics

In this section you'll find the correlation and covariance of CPI and PPI, then use a linear regression model to predict values of PPI from values of CPI.

Display the third row of the STAT menu.

[NEXT]

```
3:    [ 8.10 9.40 7.00 ]
2:    [ 2.27 5.80 1.14 ]
1:    [ 5.17 33.64 1.30 ]
  COLΣ CORR  COV   LR  PREDV
```

Here are the commands for correlation, covariance, linear regression, and predicted value.

Specifying a Pair of Columns

Before performing paired-sample statistics, specify which columns of the statistics matrix ΣDAT contain the independent and dependent data. In this example you want CPI (in column 1) to be the independent data and PPI (in column 2) to be the dependent data.

Specify columns 1 and 2 as the independent and dependent data.

1,2 COLΣ

```
3:     [ 8.10 9.40 7.00 ]
2:     [ 2.27 5.80 1.14 ]
1:   [ 5.17 33.64 1.30 ]
COLΣ | CORR |  COV  |  LR  |PREDV|
```

The numbers 1 and 2 are stored in a list named ΣPAR, which is an ordinary list with a special name. The commands that perform paired-sample statistics refer to ΣPAR.

If you don't specify the columns containing the independent and dependent data, the calculator uses columns 1 and 2. In this example you didn't need to specify the columns, but remember to execute COLΣ if your independent and dependent data aren't contained in columns 1 and 2.

Finding the Correlation

Calculate the correlation.

CORR

```
3:     [ 2.27 5.80 1.14 ]
2:   [ 5.17 33.64 1.30 ]
1:                   0.96
COLΣ | CORR |  COV  |  LR  |PREDV|
```

The correlation of CPI and PPI is 0.96.

Finding the Covariance

Calculate the sample covariance.

COV

```
3:   [ 5.17 33.64 1.30 ]
2:                   0.96
1:                  12.65
COLΣ | CORR |  COV  |  LR  |PREDV|
```

The sample covariance of CPI and PPI is 12.65.

Finding the Linear Regression

Calculate the straight line that best fits the data for CPI and PPI.

LR

```
3:                    12.65
2:                   -10.43
1:                     2.45
COLΣ CORR COV  LR  PREDV
```

The line's intercept is −10.43, and its slope is 2.45. The intercept and slope are also stored in the list ΣPAR.

Finding Predicted Values

Suppose you want to find the predicted values for PPI when CPI has values of 6 and 7. The predicted value can be calculated from the slope and intercept stored in ΣPAR.

Predict the value for PPI when CPI has value 6.

6 PREDV

```
3:                   -10.43
2:                     2.45
1:                     4.26
COLΣ CORR COV  LR  PREDV
```

The predicted value is 4.26.

Predict the value for PPI when CPI has value 7.

7 PREDV

```
3:                     2.45
2:                     4.26
1:                     6.71
COLΣ CORR COV  LR  PREDV
```

The predicted value is 6.71.

13

Binary Arithmetic

This chapter describes how to perform arithmetic with binary integers. Each binary integer contains from 1 to 64 bits and represents an unsigned binary number. For ease in entering binary numbers and reading the results, you can choose decimal, hexadecimal, octal or binary base. However, this choice doesn't affect the internal representation of binary integers, and commands act on binary integers bit-by-bit.

All commands in the BINARY menu are described briefly in appendix D, "Menu Map." For complete descriptions, refer to "BINARY" in the Reference Manual.

Selecting the Wordsize

The current wordsize affects the length of binary integers returned by commands and the display of binary integers on the stack. The wordsize can range from 1 through 64 bits, with a default wordsize of 64 bits. Suppose you want a wordsize of 16.

Before you start the example, clear the stack and display the BINARY menu.

■ CLEAR   ■ BINARY

```
3:
2:
1:
DEC■  HEX   OCT   BIN  STWS RCWS
```

Specify a wordsize of 16 bits.

16 `STWS`

```
3:
2:
1:
 DEC■│ HEX │ OCT │ BIN │STWS│RCWS
```

Now if you key in a binary integer longer than 16 bits, only the 16 least significant bits are displayed.

Selecting the Base

The current base affects how binary integers are displayed on the stack. The choices for the base are decimal, hexadecimal, octal, and binary, with a default choice of decimal base.

Suppose you want hexadecimal base.

`HEX`

```
3:
2:
1:
 DEC │ HEX■│ OCT │ BIN │STWS│RCWS
```

The label for `HEX` now includes a small square, indicating that the current base is HEX.

Entering Binary Integers

Enter the address $24FF_{16}$.

`#` 24FF `ENTER`

```
3:
2:
1:              # 24FFh
 DEC │ HEX■│ OCT │ BIN │STWS│RCWS
```

The lowercase "h" is a *base marker*, indicating that the current base is HEX. When you enter a number, you don't need to key in the base marker *unless the number is not in the current base*.

Check how this binary integer is represented in other bases. You don't need to change the binary integer, only the current mode.

Change to DEC base.

`DEC`

```
3:
2:
1:                    # 9471d
DEC■ │ HEX │ OCT │ BIN │STWS│RCWS
```

Change to OCT base.

`OCT`

```
3:
2:
1:                   # 22377o
DEC │ HEX │OCT■│ BIN │STWS│RCWS
```

Change to BIN base.

`BIN`

```
3:
2:
1:    # 10010011111111b
DEC │ HEX │ OCT │BIN■│STWS│RCWS
```

Return to HEX base.

`HEX`

```
3:
2:
1:                    # 24FFh
DEC │HEX■│ OCT │ BIN │STWS│RCWS
```

Calculating With Binary Integers

Calculate the address $1F0_{16}$ less than the given address.

`#` 1F0 `-`

```
3:
2:
1:                    # 230Fh
DEC │HEX■│ OCT │ BIN │STWS│RCWS
```

The difference is returned to level 1, just as for other numbers.

You can mix binary integers and real numbers in your calculations. A normal real integer (entered without the # delimiter) is interpreted in base 10 regardless of the current binary integer base.

For example, calculate the address 27_{10} less than the given address.

27 `-`

```
3:
2:
1:                    # 22F4h
DEC │HEX■│ OCT │ BIN │STWS│RCWS
```

The difference, expressed as a binary integer, is returned to level 1.

14

Unit Conversion

This chapter contains examples of unit conversion—converting the numerical value of a physical measurement from one system of units to another. For detailed information, refer to "UNITS" in the Reference Manual.

The UNITS Catalog

The UNITS catalog lists alphabetically all units built into the HP-28S. You'll use it to check the spelling and definition of units.

First clear the stack and select STD number display mode.

■ CLEAR ■ MODE STD

```
3:
2:
1:
STD■| FIX | SCI | ENG | DEG |RAD■
```

Start the UNITS catalog.

■ UNITS

```
 a    are
100
m^2
NEXT|PREV|         |FETCH|QUIT
```

The first unit is "are", abbreviated "a". This is a unit of area equivalent to 100 meter^2.

Try scanning forward and backward through the catalog by holding down the NEXT and PREV menu keys (not the permanent keys on the keyboard).

You can move to the first unit that begins with a particular letter by pressing that letter key.

`S`

```
s  second
1
s
NEXT PREV       FETCH QUIT
```

The entry for "second" shows that the correct abbreviation is "s" and the value is 1 second. "Second" is defined in terms of itself because it is a fundamental unit.

Be sure to use the abbreviations exactly as they appear in the UNITS catalog. For example, the HP-28S recognizes lower-case "s" as seconds, but not upper-case "S".

Next check the entry for "day."

`D`

```
d  day
86400
s
NEXT PREV       FETCH QUIT
```

This entry shows that the correct abbreviation is "d" and the value is 86,400 seconds.

Next look for the "foot" unit.

`F`

```
F  farad
1
A^2*s^4/kg*m^2
NEXT PREV       FETCH QUIT
```

The catalog shows the entry for "farad." Move forward seven entries.

`NEXT` `NEXT` `NEXT` `NEXT`
`NEXT` `NEXT` `NEXT`

```
ft  int'l foot
.3048
m
NEXT PREV       FETCH QUIT
```

The catalog shows the entry for "international foot." There are two versions of "foot" in the catalog; the next unit is "survey foot."

You can write the abbreviation for "international foot" to the command line.

`FETCH`

```
2:
1:
ft
STO  FIX  SCI  ENG  DEG  RAD
```

The normal display returns, and the command line shows the unit abbreviation.

The examples in this chapter show you how to key in units directly, but you can use ■UNITS and FETCH if you prefer.

Clear the command line.

ON

```
3:
2:
1:
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Converting Units

First convert 15 °C to degrees Fahrenheit.

Put the numerical value on the stack.

15 [ENTER]

```
3:
2:
1:                              15
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Enter the unit abbreviation for "degrees Celsius."

■[°] C [ENTER]

```
3:
2:                            .15
1:                          ' ■C '
STD■ | FIX | SCI | ENG | DEG | RAD■
```

The unit abbreviation is converted to a name.

Enter the unit abbreviation for "degrees Fahrenheit."

■[°] F [ENTER]

```
3:                            .15
2:                          ' ■C '
1:                          ' ■F '
STD■ | FIX | SCI | ENG | DEG | RAD■
```

The unit abbreviation is converted to a name.

Convert the numerical value from the old unit to the new unit.

■[CONVERT]

```
3:
2:                            .59
1:                          ' ■F '
STD■ | FIX | SCI | ENG | DEG | RAD■
```

The result shows that 15 °C converts to 59 °F.

For the next example, convert 40 inches to millimeters. This time you'll let ◼CONVERT automatically execute ENTER for you.

Clear the stack and enter the numerical value.

◼CLEAR
40 ENTER

```
3:
2:
1:                    40
STD■ FIX  SCI  ENG  DEG  RAD■
```

Enter the unit for "inches."

LC in ENTER

```
3:
2:                    40
1:                  'in'
STD■ FIX  SCI  ENG  DEG  RAD■
```

Key in the unit for "millimeter" and convert units.

You won't find "millimeter" in the UNITS catalog. It's considered a *prefixed* unit—the unit "m" (for meter) prefixed by "m" (for milli, or one-thousandth). Similarly, "km" is a prefixed unit for kilometer, and "ms" is a prefixed unit for millisecond. A complete list of prefixes appears in "UNITS" in the Reference Manual.

LC mm ◼CONVERT

```
3:
2:                  1016
1:                  'mm'
STD■ FIX  SCI  ENG  DEG  RAD■
```

The result shows that 40 inches converts to 1016 millimeters.

Converting Unit Strings

Strings are objects that contain characters. You can use *unit strings* to define more complicated units than those used so far.

A unit string can represent a unit raised to a power, such as "ft^2", or the product of units, such as "ft*lb", or any combination of unit powers and products.

A unit string can also represent a quotient of units, such as "m/sec". However, the / symbol can't appear more than once. Be sure to group all direct units before the / symbol and all inverse units after the / symbol. For example, "feet per second per second" is represented by "ft/s^2".

For the next example, convert 1 mile per hour to feet per second.

Clear the stack and enter the numerical value.

■ CLEAR
1 ENTER

```
3:
2:
1:                           1
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Enter the unit for "miles per hour."

LC mph ENTER

```
3:
2:                          1
1:                      'mph'
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Key in the unit for "feet per second."

There is no built-in unit for "feet per second," so you'll use a unit string.

■ " LC ft ÷ s

```
2:                          1
1:                      'mph'
"ft / s▮
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Alpha entry mode was activated (as indicated by the form of the cursor) when you began keying in the string. In alpha mode all commands are written to the command line, so you'll need to press ENTER to complete the string.

ENTER

```
3:                          1
2:                      'mph'
1:                  "ft / s"
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Convert the numerical value from the old units to the new units.

■ CONVERT

```
3:
2:             1.46666666667
1:                  "ft / s"
STD■ | FIX | SCI | ENG | DEG | RAD■
```

The result shows that 1 mile per hour converts to 1.46666666667 feet per second.

Next convert 10 cubic feet to gallons.

Clear the stack and enter the numerical value.

🔴 CLEAR
10 ENTER

```
3:
2:
1:              10
STD■  FIX  SCI  ENG  DEG RAD■
```

Enter the unit string for "cubic feet."

🔴 " LC ft 🔴 ^ 3 ENTER

```
3:
2:              10
1:         "ft ^ 3"
STD■  FIX  SCI  ENG  DEG RAD■
```

Key in the unit for "US gallon" and convert.

LC gal 🔴 CONVERT

```
3:
2:    74.8051948052
1:            'gal'
STD■  FIX  SCI  ENG  DEG RAD■
```

The result shows that 10 cubic feet converts to 74.8051948052 gallons.

Checking for the Correct Units

Using incorrect units can lead to unexpected numerical results or to an Inconsistent Units error. The solution in either case is to check the UNITS catalog or the "UNITS" section of the Reference Manual.

Unexpected numerical results can occur if you use a unit with the correct dimensions but an incorrect numerical value. For example, if you convert one acre to "ft^2" , the result is greater than 43,560. This occurs because there are two "foot" units, "ft" (international foot) and "ftUS" (survey foot). Converting one acre to "ftUS^2" returns exactly 43,560.

An Inconsistent Units error occurs if you use a unit with incorrect dimensions. For example, this occurs if you use "lb" (pound) as a unit of force. The correct unit for force is "lbf" (pound-force).

User Functions for Unit Conversion

If you perform particular unit conversions often, you can write user functions for those conversions. In this section you'll write user functions O→G and G→O that convert between ounces and grams; since they're user functions, you can use them in either RPN or algebraic syntax.

Recall that user functions must fulfill two requirements:

■ They must explicitly indicate their arguments.

■ They must return exactly one result.

First write O→G.

Begin the program and indicate the argument.

[«] ▮[→] [LC] x

```
2:              74.8051948052
1:                       'gal'
« → x▮
STD■ | FIX | SCI | ENG | DEG | RAD■
```

The right arrow indicates that the following name is a *local variable,* which will exist only within this program.

Define the conversion.

[«] x ['] oz ['] ['] g ['] ▮[CONVERT]
[DROP] [ENTER]

```
2:                        'gal'
1: « → x « x 'oz' 'g'
   CONVERT DROP » »
STD■ | FIX | SCI | ENG | DEG | RAD■
```

The closing delimiters are added for you.

This program means: take an argument from the stack (in RPN syntax) or from the expression (in algebraic syntax) and call it *x*; convert *x* from ounces to grams; and drop the gram unit from the stack.

Store the program in a variable O→G.

['] O ▮[→] G [STO]

```
3:
2:              74.8051948052
1:                       'gal'
STD■ | FIX | SCI | ENG | DEG | RAD■
```

Now write G→O.

Begin the program and indicate the argument.

⟦«⟧ ⬛⟦→⟧ ⟦LC⟧ x

```
2:                74.8051948052
1:                          'gal'
« → x▮
STD■  FIX   SCI   ENG   DEG  RAD■
```

Define the conversion.

⟦«⟧ x ⟦'⟧ g ⟦'⟧ ⟦'⟧ oz ⟦'⟧ ⬛⟦CONVERT⟧
⟦DROP⟧ ⟦ENTER⟧

```
2:                          'gal'
1:    « → x « x 'g' 'oz'
        CONVERT DROP » »
STD■  FIX   SCI   ENG   DEG  RAD■
```

This program means: take an argument from the stack (in RPN syntax) or from the expression (in algebraic syntax) and call it *x*; convert *x* from grams to ounces; and drop the ounce unit from the stack.

Store the program in a variable G→O.

⟦'⟧ G ⬛⟦→⟧ O ⟦STO⟧

```
3:
2:                74.8051948052
1:                          'gal'
STD■  FIX   SCI   ENG   DEG  RAD■
```

To test the conversions, check how may grams are in 1 ounce, and then convert that result back to ounces. The result should be 1 again.

Convert 1 ounce to grams.

1 ⟦USER⟧ ⟦O→G⟧

```
3:                74.8051948052
2:                          'gal'
1:                28.349523125
G→O  O→G  ΣPAR  ΣDAT   S1    PV
```

There are about 28 grams in 1 ounce. Now convert this result back to ounces.

⟦G→O⟧

```
3:                74.8051948052
2:                          'gal'
1:                             1
G→O  O→G  ΣPAR  ΣDAT   S1    PV
```

The conversions are inverses, as they should be.

15

Printing

This chapter describes some basic commands for using your HP-28S with an HP 82240A printer. Refer to the printer manual for instructions about how to position the printer relative to the HP-28S and how to turn on the printer.

All commands in the PRINT menu are described briefly in appendix D, "Menu Map." For complete descriptions, refer to "PRINT" in the Reference Manual.

Printing the Display

You can print an image of the display as follows.

1. Press and hold [ON].

2. Press [L] (the key with "PRINT" written above it).

3. Release [ON].

These keystrokes are the keyboard equivalent of the command PRLCD (*print LCD*, found on the first row of the PRINT menu). You can use these keystrokes to print the display at practically any time, without disturbing calculator operation.

If you want a program to print the display, simply include the command PRLCD, found in the PRINT menu.

Clear the stack and display the PRINT menu.

■ [CLEAR]
■ [PRINT]

```
3:
2:
1:
PR1  PRST PRVAR PRLCD  CR  TRAC
```

- PR1 (*print 1*) prints the object in level 1.
- PRST (*print stack*) prints all objects on the stack.
- PRVAR (*print variable*) prints the name and contents of a variable.
- PRLCD (*print LCD*) prints the display.
- CR (*carriage right*) prints a blank line.
- TRAC (*trace on/off*) turns Trace printing mode on and off.

Printing a Running Record

To print a running record of your calculations, turn on Trace printing mode.

TRAC

```
3:
2:
1:
PR1  PRST PRVAR PRLCD  CR  TRAC■
```

A square appears in the TRAC menu label to indicate that Trace printing mode is turned on.

Now see what happens when you add two numbers—for example, 44 and 72. First put 44 on the stack.

44 [ENTER]

```
44 ENTER
1: 44
```

The input and level 1 result are printed.

Now add 72.

72 [+]

```
72 +
1: 116
```

Again the input and level 1 result are printed.

Turn off Trace printing mode.

TRAC

```
3:
2:
1:                      116
PR1  PRST PRVAR PRLCD  CR  TRAC
```

Printing Level 1

Rather than printing all results using Trace printing mode, you can selectively print results using PR1.

PR1

```
                           116
```
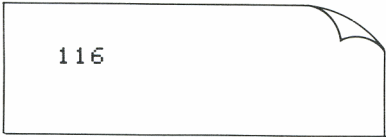
The result remains in level 1, unchanged.

You can print a message by putting a string in level 1. To print the message "OK", first put the string on the stack.

■ ["] OK [ENTER]

```
3:
2:
1:                         116
                          "OK"
 PR1  PRST PRVAR PRLCD  CR  TRAC
```

Now print the message.

PR1

```
        OK
```
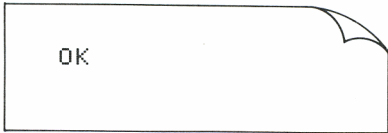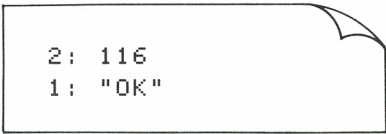
Only the contents of the string are printed, not the quotation marks.

Printing the Stack

You can print all objects on the stack by using PRST.

PRST

```
2: 116
1: "OK"
```

The contents of the stack are unchanged.

Printing a Variable

You can print the name and contents of a variable without recalling the variable to the stack. To demonstrate, store the string "OK" in a variable named "A", then print variable A.

Create the variable A with value "OK".

['] A [STO]

```
3:
2:
1:                        116
PR1  PRST PRVAR PRLCD  CR  TRAC
```

Print the name and value of the variable.

['] A PRVAR

```
A
"OK"
```

The name of the variable is dropped from the stack.

Part 2

Summary of Calculator Features

16

Objects

Part 1 of this manual contains examples of the 10 basic object types in the HP-28S. Objects are the basic entities in the calculator—the entities you create to formulate problems and manipulate to find solutions.

The purpose of most object types is to save you work by providing specific data types. For example, imagine using real numbers to represent arrays, somehow keeping track of each element in each array and writing programs to do arithmetic with these arrays. It's simpler to enter the numbers in an array object, which you can manipulate as a single entity, and to perform calculations by using the normal arithmetic functions.

However, the reason for multiple object types is broader than just multiple data types. The symbolic and programmable features of the calculator are based on symbolic objects (names and algebraics) and program objects. These objects are not just data; they can be evaluated to produce a result. (Evaluation of objects is discussed in chapter 23.)

By basing multiple data types, symbolic operations, and programming on the simple concept of object types, the HP-28S minimizes the rules you need to remember. Objects are keyed into the command line, put on the stack, or stored in variables in exactly the same way, regardless of object type.

This chapter summarizes what you learned about each object type, gives more detailed information, and suggests additional uses.

Real Numbers

Real numbers represent numbers greater than -10^{500} and less than 10^{500}. They are stored internally as a *mantissa* between 1 and 9.99999999999, a sign (positive or negative) for the mantissa, an *exponent* between 0 and 499, and a sign for the exponent.

In Hours-Minutes-Seconds Format. You can use the commands HMS+ and HMS− to add and subtract numbers expressed as hours, minutes, and seconds (or degrees, minutes, and seconds). For any computation other than addition or subtraction, first use HMS→ to convert the numbers from HMS format to decimal degree format. (See "TRIG" in the Reference Manual for details.)

Complex Numbers

Complex-number objects are ordered pairs of real numbers that represent the *real part* and the *imaginary part* of a complex number or the coordinates of a point in a plane.

Rectangular and Polar Coordinates. In chapters 7 and 8 you used complex numbers for plotting and digitizing; each complex number represented *rectangular coordinates*—that is, distances along perpendicular axes.

Chapter 6 described *polar coordinates*—a radial distance and an angle—and used the commands R→P and P→R to convert between polar and rectangular coordinates. You can use polar coordinates to key in coordinates and to display results, but you must use rectangular coordinates for calculations. The user function PSUM, described on page 86, adds points in polar coordinates by converting them, adding them, and reconverting them.

In Algebraic Objects. When you key in a complex number in an algebraic object, you may need two pairs of parentheses, as in the expression 'SIN((0,1))'. The outer pair of parentheses are required by the function SIN( ), while the inner pair are delimiters for complex numbers.

Binary Integers

Binary integers represent a sequence of bits. The length of the sequence, from 1 to 64 bits, depends on the current wordsize. The current binary integer base determines how binary integers are displayed but has no effect on their internal representation.

Large Integers. Using binary integers in decimal base mode, you can express a 19-digit positive integer exactly; this is 7 digits more than you can express exactly using real numbers.

Programming Example. The programs in "Displaying a Binary Integer," on page 257, work together to display a binary integer in all four bases.

Preserving Status. The command RCLF (*recall flags*) returns a binary integer representing the status of all 64 user flags; the command STOF (*store flags*) sets the user flags according to a binary-integer argument. These commands are demonstrated in "PRESERVE (Save and Restore Previous Status)", one of the programs in "Displaying a Binary Integer" described above.

Strings

A string comprises a sequence of characters. Part 1 showed the following uses for strings.

- In chapter 14, "Unit Conversion," you used strings to represent a combination of unit products and powers.

- In chapter 15, "Printing," you entered a message as a string in order to print it. You can also display messages by using the command DISP; it is described in chapter 27, "Interactive Programs."

Most often a string represents text, but each character can also represent a numerical value from 0 through 255. The commands CHR (*character*) and NUM (*character number*) convert between characters and their numerical values.

Non-Keyboard Characters. You can display characters that don't appear on the HP-28S keyboard by entering a numerical value and executing CHR. There are also non-displayable characters that you can print; for a list of all characters, see "STRING" in the Reference Manual.

Graphics Strings. The command LCD→ (*LCD to string*) returns a graphics string that represents the current displayed image; the command →LCD (*string to LCD*) displays the image represented by a graphics-string argument. For details about these commands, see the Reference Manual.

String Manipulations. The programs in "Displaying a Binary Integer", on page 257, show how to convert an object to string form, count the number of characters, and join two strings.

Arrays

Arrays can be one-dimensional (called *vectors*) or two-dimensional (called *matrices*), and they can comprise real or complex numbers. Chapter 11, "Vectors and Matrices," shows the basic calculations with arrays. Part 1 included the following additional uses for arrays.

- Chapter 11 shows how to solve a system of n linear equations in n unknowns by using a an n-element constant vector and an $n \times n$ coefficient matrix. For details about this process and its accuracy, see "ARRAY" in the Reference Manual.

- In chapter 12, "Statistics," the statistics data you entered was stored in the current statistics matrix ΣDAT.

In Algebraic Syntax. If an array is stored in a variable, you can refer to elements in the array by using the variable name as a function. For example, you could represent the sum of the third and fifth elements of a vector V as ' V(3)+V(5) '.

Array Manipulations. The programs in "Summary Statistics" on page 262, and "Median of Statistics Data," on page 270, demonstrate a variety of array manipulations.

Lists

Lists are sequences of objects; they are the most general method of combining several objects into one. Part 1 showed the following uses for lists.

- In chapter 4, "Repeating a Calculation," the command PATH returned a list of directory names, from the HOME directory to the current directory.

- In chapter 7, "Plotting," the list variable PPAR contained parameters used by DRAW.

- In chapter 8, "The Solver," you gave a list containing three digitized points as an estimate.

- In chapter 10, "Calculus," you specified the variable of integration and the lower and upper limits of integration by combining them in a list.

- In chapter 12, "Statistics," the list variable ΣPAR contained parameters for paired-sample statistics.

In Algebraic Syntax. If a list is stored in a variable, you can refer to elements in the list by using the variable name as a function. For example, you could represent the sum of the third and fifth elements of a list L as 'L(3)+L(5)'.

Lists and the Stack. The program MEDIAN, on page 273, shows how to put the elements of a list on the stack and combine objects on the stack into a list.

Sorting a List. The program SORT, on page 270, shows how to sort the elements in a list.

Extracting Elements From a List. The program LMED, on page 272, shows how to extract elements from a list.

Names

Names are a sequence of characters used to name other objects. They can contain up to 127 characters, although practical considerations suggest that names be no longer than five or six characters.

The legal characters available on the keyboard are letters, digits, and the characters ? Σ π → μ ▪ . The first character can't be a digit. The following characters cannot be included in names.

- Characters that separate objects: delimiters (# [ ] " ' { } ‹ › « »), space, period, or comma.
- Algebraic operator symbols (+ − * / ^ √ = < > ≤ ≥ ≠ ∂ ∫)

The calculator determines whether a name is global or local when the command line is processed: if the name is used by a program structure to create a local variable, the name is local within that program structure; otherwise, the name is global.

Local Names. In part 1 you wrote user functions that created local variables. This manual used lowercase letters for the local names to help you distinguish them from global names. It's important to remember that it was the command → that made the names local, not the lowercase letters. If you name a local variable e or i, your local definition supersedes the built-in definition.

Global Names. All the other names in part 1 were global. Examples include:

- Names for global variables (numerical variables used for plotting or the Solver; all variables in the USER menu).
- Names for directories.
- Names used symbolically, without reference to specific values (symbolic arithmetic, symbolic solutions, and calculus).

Names of commands, including e, i, and π, can't be used as global names. In addition, the following names are reserved for specific uses.

- EQ refers to the current equation used by the Solver and PLOT commands.

- ΣPAR refers to a list of parameters used by statistics commands.

- PPAR refers to a list of parameters used by plot commands.

- ΣDAT refers to the current statistical array.

- s1, s2, and so on, are created by ISOL and QUAD to represent arbitrary signs obtained in symbolic solutions.

- n1, n2, and so on, are created by ISOL to represent arbitrary integers obtained in symbolic solutions.

- Names beginning with "der" refer to user-defined derivatives.

You can use any of these names for your own purposes, but remember that certain commands use these names as implicit arguments.

Programs

Programs are sequences of objects and commands. Each program is essentially a command line made into an object; when you surround the contents of the command line by program delimiters, you indicate that you want to save the contents for later execution.

Special program commands appear in the PROGRAM BRANCH, PROGRAM CONTROL, and PROGRAM TEST menus. These menus are described in the Reference Manual, along with the general topic "Programs."

You wrote five programs in part 1:

- In chapter 3 you wrote a program for renaming variables, and you stored it in the variable RENAME.

- In chapter 5 you wrote a program for the cotangent function, and you stored it in the variable COT.

- In chapter 6 you wrote a program for adding polar coordinates, and you stored it in the variable PSUM.

- In chapter 14 you wrote programs for converting between ounces and grams, and you stored them in the variables O→G and G→O.

User Functions. The programs COT, PSUM, O→G, and G→O are user functions—they begin with the command → and one or more names, which together define one or more local variables, followed by one expression or program. When the user function is stored in a variable, you can use the name of the variable in algebraics as you would use a built-in function.

Program Structures. The command → followed by names and an expression or program is called a *local-variable structure*, which is one type of program structure. There are also program structures for branching (such as IF ... THEN ... ELSE ... END) and looping (such as DO ... UNTIL ... END). See chapter 26, "Program Structures," for descriptions. Also, chapter 28, "Programming Examples," contains 20 programs that demonstrate every program structure, along with a variety of programming techniques.

Unnamed Programs. Programs don't need to be stored in variables to be useful; for examples, see "Expanding and Collecting Completely," on page 253, and "Displaying a Binary Integer," on page 257.

Algebraics

Algebraics comprise one or more functions and the functions' arguments; the arguments can be numbers, names, or subexpressions. Algebraics are written and displayed in algebraic syntax, a form similar to written mathematical notation. There are two types of algebraics, expressions and equations.

Expressions

In part 1 you used expressions in three different ways: as data, as functions, and as implicit equations.

Expressions As Data. When you calculate with expressions, such as adding two expressions, squaring an expression, or differentiating an expression, the result is another expression. In these cases the expressions act as data to be manipulated, independent of any values assigned to the variables.

Expressions as Functions. In chapter 4 you created the expression RTOT and, using the Solver, assigned values to the variables and then evaluated RTOT to calculate the desired result. In this case the expression acted as a function which, given the input values, produced a result.

Expressions as Implicit Equations. In chapter 8 you used the Solver to find the *numerical* zero of an expression—that is, the numerical value of the independent variable for which the expression has value 0. In chapter 9 you used QUAD to find a *symbolic* zero—that is, an expression which, substituted for the independent variable, would give the original expression the value 0.

In both cases the expression $f(x)$ acts like the equation $f(x) = 0$, because the *zero* of the expression is the same as the *root* of the equation.

Equations

Equations comprise two expressions related by an equals sign ($=$). In mathematics there are two uses for the equals sign:

- To indicate a proposition, such as "$x^2 = 4$" or "$x^2 + y^2 = 1$." Here the equation holds only for some values of the variables.

- To indicate an identity or definition, such as "$\sin 2x = 2 \sin x \cos x$" or "$y = 3x^2 + 2x + 5$." Here the equation holds for all values of the variables.

On the HP-28S, equations are used for propositions only; to make a definition such as "$y = 3x^2 + 2x + 5$," the expression `'3*X^2+2*X+5'` is *stored in a variable* named Y.

In "Time Value of Money" on page 103, both TVM and SPPV are expressed mathematically as equations. The TVM equation, which holds only for certain values of its variables, is entered as an equation; but SPPV, whose value is defined by the value of its variables, is created as a variable.

Equations as Data. When you calculate with equations, such as adding two equations, or squaring an equation, or differentiating an equation, the result is another equation. Each side of the equation is treated independently—each side is an expression treated as data. The equation maintains its propositional nature, where it holds for only some values of its variables.

Solving Equations. When you solve an equation numerically, as you did in "Time Value of Money," you find the value of the independent variable that satisfies the equality. Similarly, when you solve an equation symbolically, as you did in "Isolating a Variable" on page 109, you find an expression which, substituted for the independent variable, would satisfy the equation.

Symbolic Constants

Algebraics can include the following symbolic constants. These look like names but are actually functions.

- MINR (*minimum real*) represents the smallest positive real number. Its numerical value is $1.00000000000\text{E}-499$.

- MAXR (*maximum real*) represents the largest positive real number. Its numerical value is $9.99999999999\text{E}499$.

- e represents the base of natural logarithms. Its numerical value on the HP-28S is 2.71828182846.

- π represents the ratio of circumference to diameter of a circle. Its numerical value on the HP-28S is 3.14159265359.

- i represents the imaginary number $\sqrt{-1}$. Its numerical value is $(0, 1)$.

In Numerical Constants mode or Numerical Result mode, evaluation of symbolic constants returns their numerical values; otherwise, evaluation returns their symbolic form. (Constants mode and Result mode are described in chapter 24.)

17

Operations, Commands, and Functions

Each procedure built into the HP-28S can be classified as an operation, a command, a function, or an analytic function.

- An *operation* is any procedure built into the calculator.

- A *command* is a programmable operation.

- A *function* is a command allowed in algebraics.

- An *analytic function* is a function for which the HP-28S provides a derivative and inverse.

Built-in procedures are usually characterized by their highest capability. For example, SWAP and IP are both commands, but we characterize SWAP as a command and IP as a function. The following table shows examples of each type.

Operations

Non-Programmable Operations	Commands		
	RPN Commands	Functions	
		Non-Analytic	Analytic
INS	SWAP	ABS	ASIN
NEXT	DROP	∂	EXP
EDIT	LAST	IP	INV
VIEW↑	RCL	MAX	LN
ENTER	PURGE	OR	NEG
EEX	∫	%CH	SIN
COMMAND	STO	R→D	SINH
UNDO	EVAL	R→P	SQ
CONT	CLEAR	XPON	+
ON	CONVERT	≠	=

The Operation Index in the back of the Reference Manual identifies each built-in procedure as an operation, a command, a function, or an analytic function. As a rough guide, here are general comments about each type.

■ Most non-programmable operations can be executed only by pressing a key. However, there are programmable equivalents for some operations: for example, the [TRIG] operation (to select the TRIG menu) can be effected in a program by executing 21 MENU, and the RAD operation (to select Radians angle mode) can be effected by executing 60 FS.

■ Most RPN commands involve manipulating the stack or altering user memory rather than calculating mathematical values.

■ Most non-analytic functions are mathematical calculations without inverses—that is, they return some characteristic of the arguments, but the arguments can't be reconstructed from the result. Examples include integer part and fractional part, absolute value and sign.

■ In mathematics, a function of complex variables is *analytic* if it can be expressed as a power series at every point in its domain; in this case it has an inverse and a derivative. The HP-28S makes a few exceptions to this definition. For example, no derivative is given for the command %, although one would be possible; a derivative *is* given for the function ABS, although the function is non-analytic at the point $0 + 0i$.

Every built-in procedure is available on a key, either on the keyboard or in a menu. When you press a key, the exact result depends on the type of procedure and the *entry mode*, as discussed in the next chapter.

18

The Command Line

The command line holds any number of characters representing objects in text form. It appears at the bottom of the display (immediately above the menu labels, if present) when you begin to key in an object or when you use ▮EDIT or ▮VISIT to edit the contents of an existing object.

The command line can hold more than one row of text. If you enter more than 23 characters into one row, characters scroll off the display to the left. An ellipsis (...) appears in the leftmost character position to indicate the undisplayed characters. If you try to move the cursor past the left end of the display, the leftmost characters scroll back into the display, and characters scroll off the display to the right. An ellipsis then appears at the right end of the display. When the command line contains multiple rows of text, all rows scroll left and right together.

The Cursor Menu

The cursor menu is a special menu of editing operations. It is active whenever the command line is present and no menu labels are visible. The cursor menu contains both shifted and unshifted keys. The unshifted keys are labeled in white above the corresponding menu keys, as illustrated.
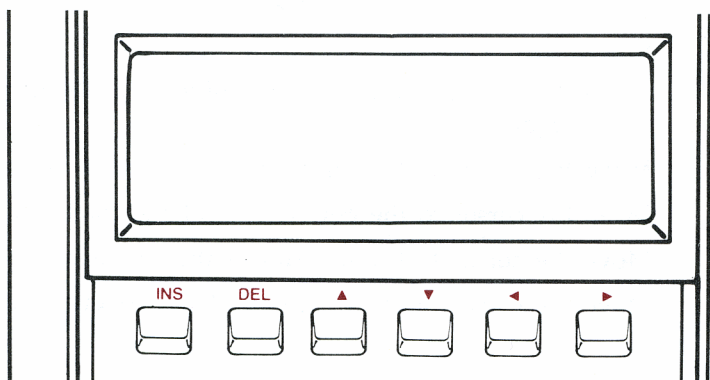
If you press and hold an unshifted cursor menu key (except [INS]), the operation is repeated until you release the key.

Key	Description
[INS]	Switch between Replace mode and Insert mode. In Replace mode, new characters replace existing characters. In Insert mode, new characters are inserted between existing characters.
[DEL]	Delete the character at the cursor position.
[▲]	Move the cursor up one line.
[▼]	Move the cursor down one line.
[◄]	Move the cursor left one space.
[►]	Move the cursor right one space.

The shifted cursor menu keys (except for ■[INS]) are equivalent to repetitions of the unshifted operations.

Key	Description
■ INS	Delete all characters to the left of the cursor.
■ DEL	Delete the character at the cursor position and all characters to the right.
■ ▲	Move the cursor to the top row of the command line.
■ ▼	Move the cursor to the bottom row of the command line.
■ ◄	Move the cursor to the left end of the command line.
■ ►	Move the cursor to the right end of the command line.

Some Entry Keys

The following keys are useful when you're entering objects in the command line.

Key	Description
◄♦►	**Cursor Menu On/Off.** When the Cursor menu is not active: selects the Cursor menu. When the Cursor menu is active: selects the previous menu.
CHS	**Change Sign.** When the cursor is positioned at a number: changes the sign of the number. When the cursor is not positioned at a number: writes a minus sign. (If no command line is present: executes the command NEG.)
EEX	**Enter Exponent.** When the cursor is positioned at a number without an exponent: writes the character E after the number. When the cursor is positioned at a number with an exponent: positions the cursor after the E. If the cursor is not positioned at a number: writes 1E.
◄	**Backspace.** Deletes the character to the left of the cursor, moving the cursor (and any characters to the right) one space to the left. If you press and hold ◄, the action is repeated until you release the key.
LC	**Lower-Case Letters.** Switches between Upper-case and Lower-case modes. When the command line is created, Upper-case mode is active—[A] through [Z] write A through Z. In Lower-case mode, [A] through [Z] write a through z.

| MENUS | **Menu Lock.** Turns Menu Lock on and off. When Menu Lock is on, the shifted and unshifted "positions" are switched for the top three rows of the lefthand keyboard (letter keys [A] through [R]). You don't need to press ▓ before [ARRAY] through [UNITS], but you need to press ▓ before the letters A through R. |

| ON | **Attention.** Cancels the command line. |

Object Delimiters and Separators

To enter more than one object or command into the same command line, you must separate them by one of the following:

■ An object delimiter: ( ) [ ] { } # " ' « ».

■ A space or newline. Pressing ▓ NEWLINE inserts a "newline" character (line-feed) into the command line at the cursor position. Newline characters are equivalent to spaces when the command line is executed.

■ A comma (assuming you haven't selected the comma to act as the decimal point).

Entry Modes

To make object entry easier, there are three entry modes—*Immediate*, *Algebraic*, and *Alpha*—for entering different types of objects. Remember the distinctions made in the previous chapter, "Operations, Commands, and Functions":

■ *Operations* are not programmable.

■ *Commands* can appear in programs but not in algebraics.

■ *Functions* (analytic and non-analytic) and *names* can appear in programs or algebraics.

The calculator recognizes these distinctions as you enter objects in the command line. Pressing an operation key (such as [ENTER]) always causes execution of the operation. The current entry mode primarily affects what happens when you press a command key (such as [STO]), a function key (such as [+]), or a USER menu key.

Immediate Entry Mode. This mode is for entering numbers, lists, and arrays. In Immediate entry mode:

- Pressing a command key executes the command line and then executes the command.

- Pressing a function key executes the command line and then executes the function.

- Pressing a USER menu key executes the command line and then evaluates the corresponding name.

Algebraic Entry Mode. This mode is for entering names and algebraics. If you begin the command line by pressing ['], Algebraic entry mode is automatically activated. In this mode:

- Pressing a command key executes the command line and then executes the command.

- Pressing a function key writes the function's name in the command line. If the function takes its arguments in parentheses, the opening parenthesis is included.

- Pressing a USER menu key writes the corresponding name in the command line.

Alpha Entry Mode. This mode is for entering strings and programs. Pressing ▮["] or [«] automatically activates Alpha entry mode and turns on the *α* annunciator. In this mode:

- Pressing a command key writes the command's name in the command line.

- Pressing a function key writes the function's name in the command line.

- Pressing a USER menu key writes the corresponding name in the command line.

If the cursor is positioned at the end of the command line, or if Insert mode is active, spaces are included as needed to keep successive commands separate.

Exceptions

To enable you to select a mode while using the command line in Immediate or Algebraic entry mode, the following command keys execute their command without disturbing the command line.

- `STD` , `DEG` , and `RAD` in the MODE menu.
- `DEC` , `HEX` , `OCT` , and `BIN` in the BINARY menu.

Since the following commands make sense only in a program, pressing one of these keys always writes the command's name in the command line.

- `HALT` in the PROGRAM CONTROL menu.
- All keys in the PROGRAM BRANCH menu.

To help prevent the accidental loss of variables, pressing `CLUSR` (in the MEMORY menu) always writes CLUSR in the command line. You must then press [ENTER] to execute the command.

Manual Selection of Entry Modes

The calculator automatically switches between Immediate and Algebraic entry modes each time you press ['] to begin or end a name or algebraic. It also switches to Alpha entry mode when you press ■["] or [«]. You can manually select the entry mode by pressing the [α] key. Doing so switches the entry mode in the cycle illustrated below.

Immediate ⟶ Alpha ⟶ Algebraic
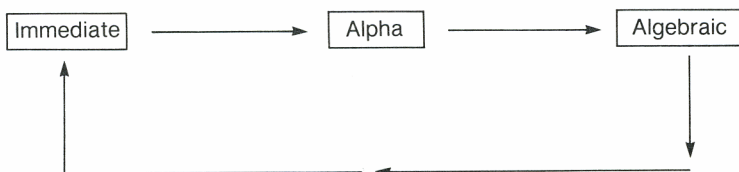
Manual Selection of Entry Modes

Thus you can switch to any entry mode by pressing $\boxed{\alpha}$ once or twice. Here are some examples of using the $\boxed{\alpha}$ key.

■ Suppose you want to write a program that you'll execute only once or twice. Press $\boxed{\alpha}$ to select Alpha entry mode; key in the program without program delimiters; press $\boxed{\text{ENTER}}$ to execute the program; press ■ $\boxed{\text{COMMAND}}$ to return the program to the command line; press $\boxed{\text{ENTER}}$ to execute the program again.

■ Suppose you want to purge several variables at once. Press $\boxed{\{\,\}}$ to start a list; press $\boxed{\alpha}$ to select Alpha entry mode; press the USER menu keys for the variables to be purged; press $\boxed{\text{ENTER}}$ to put the list on the stack; press ■ $\boxed{\text{PURGE}}$.

■ Suppose you're keying in a program, and you want to use the character → in a name. Since Alpha entry mode is active, pressing ■ $\boxed{\rightarrow}$ would write the command "→" surrounded by spaces. Press $\boxed{\alpha}$ to select Algebraic entry mode; press ■ $\boxed{\rightarrow}$; press $\boxed{\alpha}$ $\boxed{\alpha}$ to return to Alpha entry mode.

How the Cursor Indicates Modes

The appearance of the cursor indicates the current entry mode and the current choice of Insert or Replace mode. The following table shows the six possible combinations of entry mode and Insert or Replace mode.

	Insert mode	Replace mode
Immediate entry mode	◁	□
Algebraic entry mode	⇔	⊟
Alpha entry mode	◀	▦

Executing the Command Line

When you press $\boxed{\text{ENTER}}$ (or a key that performs ENTER in the current entry mode), the calculator does the following:

1. The busy annunciator ((•)) is turned on.

2. If UNDO is enabled, a copy of the current stack is saved.

3. The text string in the command line is searched for object delimiters and separators, then broken into the corresponding substrings.

4. Each substring of text is tested against syntax rules to identify its object type.

5. If COMMAND is enabled, a copy of the command line is saved in the command stack.

6. The command line is executed.

7. The busy annunciator ((•)) is turned off.

If a substring fails the syntax tests in step 4, steps 5 and 6 are not performed. Instead, Syntax Error is displayed, and the incorrect text is highlighted in inverse characters, followed by the cursor. If the error resulted from incomplete syntax, the cursor is positioned at the end of the line.

Editing Existing Objects

You can return an existing object to the command line, view it or edit it using command-line operations, and replace the original object with the modified object if desired.

Key	Description
■ $\boxed{\text{EDIT}}$	**Edit Level 1.** Returns the object in level 1 to the command line.
n ■ $\boxed{\text{VISIT}}$	**Edit Level n.** Returns the object in level n to the command line.
'name' ■ $\boxed{\text{VISIT}}$	**Edit a Variable.** Returns the contents of the specified variable to the command line.

The cursor menu and Alpha entry mode are activated. The original object, if visible, is highlighted to remind you that you are editing that object and that the original copy is still preserved.

When you're done viewing or editing the object, you can:

■ Press [ON] to cancel the edit, clear the command line, and leave the original object unchanged.

■ Press [ENTER] (or a key that performs ENTER) to replace the original object.

If the cursor menu is still active when you complete the editing, the previous menu is restored.

Recovering Command Lines

The HP-28S saves the contents of the last four command lines you executed. Pressing ■ [COMMAND] once returns the most recently executed command line (replacing the current command line if it exists); pressing ■ [COMMAND] a second time returns the next oldest command line; and so on. If you press ■ [COMMAND] more than four times, the sequence starts over with the most recent command line.

Some uses for ■ [COMMAND] appear in "If You Execute the Wrong Function" on page 47 and "Manual Selection of Entry Modes" on page 171.

You can disable this feature by pressing CMD in the MODE menu. The box disappears from the menu label, indicating that command lines won't be saved. To enable this feature again, press CMD a second time.

The Command Line as a String

The text that you key into the command line is equivalent to the contents of a string object—that is, a sequence of characters. You can programmatically execute a command line by entering the text in a string and executing STR→ (*string-to-objects*). This technique is useful for storing programs in text form, which is more compact than object form. Also, any local names that exist when STR→ is executed will be recognized in the command line.

19

The Stack

This chapter reviews what you've learned about the stack and describes commands for manipulating objects on the stack. Also briefly described is the use of local variables to simplify stack manipulations.

Review of Stack Concepts

The stack is a sequence of numbered *levels*, each holding one object. The objects you key into the command line are put on the stack when you execute ENTER. The first object in the command line is the first object put on the stack. Each object is put in level 1, lifting other objects to the next higher stack level. The stack can grow indefinitely (within the limits of calculator memory), so you don't need to think about how many objects are on the stack before entering more objects.

In general, a command removes input objects (called *arguments*) from the stack and replaces them with output objects (called *results*) to the stack. For example, the function + removes two arguments from levels 1 and 2, replacing them with their sum in level 1.

The arguments must be present on the stack before the command is executed. This type of logic, where the command comes after the arguments, is called *stack logic*, *postfix logic*, or *RPN*, for *Reverse Polish Notation*, in honor of the Polish logician Jan Łukasiewicz (1878-1956).

The results of a command are available as arguments for the next command. If you're not ready to use the results yet, simply leave them on the stack—they'll be available when you're ready for them.

Objects leave the stack from level 1, and the objects remaining on the stack each drop to a lower level. It's best to drop an object or store it in a variable when you don't need it on the stack; this makes it easier to keep track of the stack objects you do need. Similarly, it's best to clear the entire stack when you begin a problem, so you'll know that the objects on the stack are pertinent.

Viewing the Stack

Normally you see only the first few objects on the stack. If the object in level 1 is large, you see only the first part of it. The operations ■ VIEW↑ and ■ VIEW↓ enable you to view the first part of any object on the stack and all of the object in level 1.

These operations move the "window" through which you see the stack. The size of this window can range from one to four display lines, depending on the presence of a menu, the command line, or both.

Key	Description
■ VIEW↑	Moves the window up (toward higher stack levels).
■ VIEW↓	Moves the window down (toward the end of the object in level 1).

Viewing has no effect on the contents of the stack, the command line, or the action of commands.

Manipulating the Stack

In part 1 you used some basic commands for manipulating the stack: CLEAR (to clear the stack), DROP (to drop the object in level 1), and SWAP (to switch the order of the objects in levels 1 and 2). This section briefly describes all commands for moving, copying, and dropping stack objects; for details, see "STACK" in Reference Manual.

Moving Stack Objects. These commands rearrange the objects on the stack; the number of objects doesn't change. Commands preceded by "*n*" require a real-number argument.

Command	Description
SWAP	Moves the object in level 2 to level 1.
ROT	Moves the object in level 3 to level 1.
n ROLL	Moves the object in level *n* to level 1.
n ROLLD	Moves the object in level 1 to level *n*.

The command names ROT (*rotate*), ROLL, and ROLLD (*roll down*) are descriptive of the motion of objects as a block. ROT moves the object in level 3 to level 1, rotating a block of three objects; ROLL and ROLLD roll blocks of *n* objects.

Copying Stack Objects. These commands return a copy of one or more stack objects. Copying only one object returns the copy to level 1 and lifts the other objects on the stack (including the original object) to a higher level. When you copy more than one object, they're copied as a block in a similar manner. Commands preceded by "*n*" require a real-number argument.

Command	Description
DUP	Copy the object in level 1. (When no command line is present, you can execute DUP by pressing ENTER.)
OVER	Copy the object in level 2.
n PICK	Copy the object in level *n*.
DUP2	Copy the objects in levels 1 and 2.
n DUPN	Copy the objects in levels 1 through *n*.

Dropping Stack Objects. These commands drop one or more objects from the stack. The objects remaining on the stack are dropped to a lower level. Commands preceded by "*n*" require a real-number argument.

Command	Description
DROP	Drop the object in level 1.
DROP2	Drop the objects in levels 1 and 2.
n DROPN	Drop the objects in levels 1 through *n*.
CLEAR	Drop all objects.

Local Variables

In part 1 you wrote a few user functions—programs that define local variables and use them in a single expression or program. User functions can be included in algebraics, just like built-in functions.

The use of local variables reduces the need for stack manipulations. When you create local variables, their values are removed from the stack. You can then refer to them by name instead of finding them on the stack.

Local variables have applications in addition to user functions. Almost all of the programming examples in chapter 28 use local variables. Of particular interest are "Box Functions" on page 241, "MULTI (Multiple Execution)" on page 253, "PRESERVE (Save and Restore Previous Status)" on page 258, and "SORT (Sort a List)" on page 270.

Recovering the Last Arguments

The HP-28S saves the arguments to the last command executed. Depending on the command, one, two, or three objects may be saved. (If a command takes no arguments, the previous saved arguments are preserved.) The command LAST returns the saved arguments, each to the stack level it occupied originally.

If you need exactly the same arguments for two or more commands in sequence, you can execute LAST to return copies of the arguments to the stack for the next command. If the commands don't require exactly the same arguments, or if the commands aren't in sequence, it's easier to use local variables.

You can disable LAST (that is, the saving of arguments) by pressing `LAST` in the MODE menu. The box disappears from the menu label, indicating that arguments won't be saved. This practice is not generally recommended, since the calculator uses the saved arguments for recovery when an error occurs. However, if a command or program fails because of insufficient memory, you can attempt execution with LAST disabled. When you're done, be sure to enable LAST again by pressing `LAST` a second time.

Restoring the Stack

Each time you press ENTER (or a key that performs ENTER) the HP-28S first saves a copy of the stack and then performs the specified actions. If you're dissatisfied with the results, you can restore the saved stack by pressing ■ UNDO. Note that UNDO affects only the stack—it doesn't undo changes to user flags or user variables. For an example using ■ UNDO, see "If You Execute the Wrong Function" on page 47.

You can disable this feature by pressing `UNDO` in the MODE menu. The box disappears from the menu label, indicating that the stack won't be saved. To enable this feature again, press `UNDO` a second time.

The Stack as a List

The contents of the stack are equivalent to the contents of a list—that is, a sequence of objects. You can put all of the objects on the stack into a single list by executing DEPTH →LIST. The command DEPTH returns the number of objects on the stack, and the command →LIST (*stack to list*) combines the specified number of objects into a list.

More often, a list is "opened" onto the stack by the command LIST→ (*list to stack*). After the elements are manipulated on the stack, they may be recombined into a list by the command →LIST. For examples of these commands, see "MEDIAN (Median of Statistics Data)" on page 273.

20

Memory

Memory is used for a variety of purposes in the HP-28S, including the command line, the stack, user memory, recovery features, and the operating system. The command line and the stack are described in chapters 18 and 19. This chapter primarily discusses user memory, including directories; it also discusses low-memory conditions and its effects on recovery features and the operating system.

User Memory

User memory can contain variables, and it can contain directories to organize the variables.

Global Variables

A variable is the combination of a name object and any other object. The name object represents the name of the variable; the other object is the value or contents of the variable.

Global variables are those that are stored in user memory. There are also *local* variables, which are created by program structures and exist only during execution of the program structures. Local variables are primarily a substitute for stack manipulations and are described in chapter 19, "The Stack." In the present chapter, the term "variables" indicates global variables.

The contents of a variable can be any type of object. In part 1 you created numerical variables, program variables, algebraic variables, list variables, and array variables. You even created name variables, where the contents of the variable was the name of another variable.

You used the following commands to create, recall, and purge variables. These commands treat all variables alike, regardless of their contents.

Command	Description
STO	Creates a variable with the specified value and name.
RCL	Recalls the contents of the specified variable.
PURGE	Deletes one or more specified variables.

Directories

In chapter 4, "Repeating a Calculation," you used the Solver to calculate the total resistance of two series-parallel circuits, with two sets of resistor values that could be applied to either circuit. Here is a review of the concepts you learned.

There are two primary motivations for creating directories.

- To *group together* the variables for a particular application or topic. You created the directory EE for your electrical engineering problems so that, when you're working on these problems, you can focus on the relevant variables. Equally important, when you're working on other problems, the electrical engineering variables are all hidden within the EE directory.

- To *keep separate* sets of variables that use the same names. You created directories SP1 and SP2 (*series-parallel-1* and *series-parallel-2*) within EE to hold different values of the variables R1, R2, and R3. You can switch from one set of values to the other simply by switching directories.

Creating a Directory. To create a directory you enter a name and execute CRDIR (*create directory*). The name of the directory appears in the USER menu. The new directory is called a *subdirectory*, and the directory that contains it is called its *parent directory*.

The Current Directory. Initially, the only directory that exists is the built-in directory HOME. After creating other directories, you can choose which is the *current directory*—that is, which set of variables appears in the USER menu.

To choose the current directory you evaluate its name—for example, if you've just created a directory, you make it the current directory by pressing the appropriate key in the USER menu.

Almost all commands that use variables work only in the current directory, since the purpose of multiple directories is to control which variables are available. You can alter a variable only if it's in the current directory.

The following commands in the MEMORY menu act on the current directory.

Command	Description
VARS	Returns a list of names of all variables and directories in the current directory.
ORDER	Reorders variables and directories in the current directory as specified by a list.
CLUSR	Purges all variables and empty directories in the current directory.

The Current Path. You can check where you are in the directory structure by executing the command PATH. It returns a list specifying the sequence of directories from the HOME directory to the current directory.

In some cases the calculator searches not only the current directory, but the entire current path. The search begins in the current directory; if the variable isn't found, the search continues in the parent directory; and this process continues back to the HOME directory.

This occurs in the evaluation of names—after all, you could never return to a parent directory if you couldn't successfully evaluate its name. Evaluation of names occurs when you key in an unquoted name, when you plot or use the Solver, when you evaluate algebraics on the stack, and so on.

Other commands that search the current path are RCL and PRVAR (*print variables*). Note that none of the actions that search the current path can alter the variable.

Since the HOME directory is always on the current path, the calculator can always find variables in the HOME directory. You might choose to limit the contents of the HOME directory to subdirectories and those variables you want always available.

Directory Structure. The diagrams below show the directory structure you created in chapter 4. In the first diagram, HOME is the current directory; in the second, EE; and in the third, SP2. Each diagram uses the following symbols.

Symbols Used in the Directory Diagrams

name   Name of a directory.

name   A name in the current directory. These names appear in the USER menu. The corresponding variables can be altered.

_____   The current path.

name   A name on the current path. These names can be found only by evaluation, RCL, and PRVARS. The corresponding variables can't be altered.
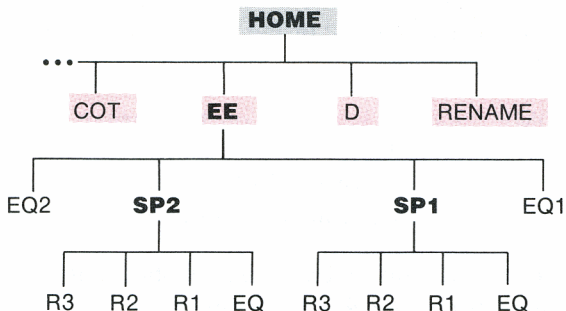
Current Directory is HOME
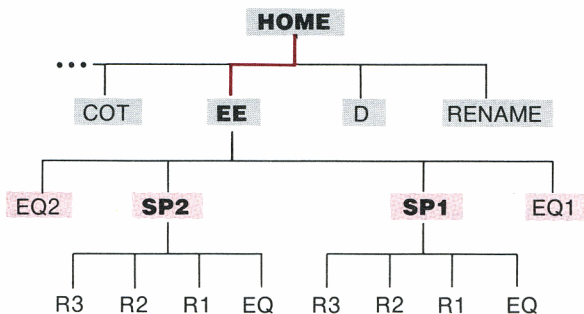
Current Directory is EE
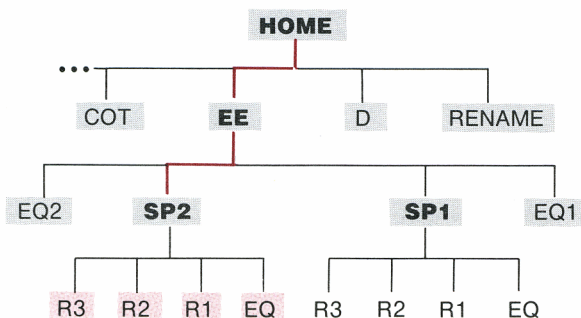
Current Directory is SP2.

Purging a Directory. You can purge an *empty* directory just as you would a variable: switch to the directory that contains the directory to be purged, put the directory's name on the stack, and execute PURGE.

If the directory to be purged contains variables or subdirectories, you must purge the variables or subdirectories before you can purge the directory. Here's a general procedure.

1. Switch to the directory to be purged.
2. Execute CLUSR to clear the directory.
3. Switch to the parent directory.
4. Purge the directory.

If a Non-Empty Directory error occurs in step 2, the directory contains a subdirectory that isn't empty. In this case you must perform steps 1, 2, and 3 to clear the subdirectory. You can then continue with step 2 through 4 to purge the directory.

Moving Up and Down the Directory Structure. Chapter 28 includes programs for moving up the directory structure (switching to a parent directory) or moving down (switching to a subdirectory). See "Changing Directories" on page 275.

Recovery Features

The HP-28S automatically saves copies of command lines, arguments, and the stack. These copies enable you to recover from a mistake—to go back to where you were before the mistake. You can then redo a calculation without starting over from the beginning. The copies of command lines and arguments are also handy for repeating calculations.

These copies can consume a significant amount of memory. For each of these recovery features—command lines, the stack, and arguments—you can choose whether to enable or disable the feature. The operations to enable or disable the recovery features appear in the MODE menu.

Generally it's best to leave these features enabled. If very little memory is available and large objects have been saved by the recovery features, you can safely regain some memory by disabling and re-enabling each feature, thereby clearing the stored objects.

Low Memory

The HP-28S contains 32 Kbytes of user memory, of which about 400 bytes are reserved for system use. Virtually every HP-28S operation requires some memory use—even interpreting the command line. The amount of memory used by some algebra commands (COLCT, EXPAN, TAYLR) increases rapidly as their arguments become more complicated. Try to leave at least a few thousand bytes of memory free for dynamic system use.

You can check the amount of available memory by executing MEM, found in the MEMORY menu.

Because the HP-28S operating system shares memory with user objects, you can fill memory so full of user objects that normal calculator operation becomes difficult or impossible. The HP-28S provides a series of low memory warnings and responses, listed below in order of increasing severity.

Insufficient Memory. If there isn't enough memory available for a command to execute, an Insufficient Memory error occurs. If LAST is enabled, the original arguments are restored to the stack. If LAST is disabled, the arguments are lost.

No Room for UNDO. If there isn't enough memory available to save a copy of the stack, a No Room for UNDO error occurs. The UNDO feature is automatically disabled; to reenable UNDO, press UNDO in the MODE menu.

No Room to ENTER. If there isn't enough memory available to process the command line, the calculator clears the command line and displays No Room to ENTER. A copy of the unsuccessful command line is saved in the command stack if the command stack is enabled.

If you're attempting to edit an existing object, using EDIT or VISIT, and a copy of the unsuccessful command line is saved in the command stack, purge the original copy of the object, press ■ COMMAND to recover the command line containing the edited object, and press ENTER to enter the edited version.

Low Memory! If fewer than 128 bytes of free memory remain, Low Memory! flashes once in the top line of the display. This message will flash at every keystroke until additional memory is available. Clear unneeded objects from memory before continuing your calculations.

No Room To Show Stack. It is sometimes possible for the HP-28S to complete all pending operations, and not have enough free memory left for the normal stack display. 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—algebraics usually require the most memory. Clear one or more objects from memory, or store a stack object as a variable so that it does not 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, build a command line, and so on. In this situation, you *must* clear some memory before continuing. A special Out of Memory procedure is activated, which will create a display:

```
Out of Memory
Purge?
Command Stack
 YES   NO
```

The calculator will sequentially prompt you to clear:

1. The COMMAND stack (if enabled).
2. The UNDO stack (if enabled).
3. LAST Arguments (if enabled).
4. The custom menu (if any).
5. The stack.
6. Each variable in the HOME directory.

For each item that you want to purge, press the `YES` menu key; for those that you want to keep, press `NO`.

After pressing `YES` at least once, you can try to terminate the Out of Memory procedure by pressing `ON`. If sufficient memory is available, the calculator returns to the normal display; otherwise, the calculator beeps and continues through the purge sequence. After cycling once through the choices, the Out of Memory procedure 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.

If you press `YES` for an empty directory, it is purged. If you press `YES` for a directory that contains variables, the variables in that directory are displayed.

Maximizing Performance

From time to time the calculator does "housekeeping" to make better use of memory. Generally this process is noticeable only as short pauses during plotting, for example; however, when memory is almost full and the stack contains hundreds of objects, the calculator may respond slowly to even simple operations such as selecting a menu.

This section contains tips for maximizing speed (by reducing the amount of housekeeping required) and maximizing available memory (by increasing the effectiveness of housekeeping).

To Maximize Speed:

- Don't put more than a few hundred objects on the stack.
- Don't leave large lists (more than a few hundred objects) on the stack; store them in user memory.

To Maximize Available Memory:

> **Note** The following procedure clears the stack, recovery data (COMMAND, UNDO, LAST), the current custom menu (CUSTOM), and any suspended programs.

1. Purge unwanted variables and directories from user memory.
2. Store in user memory any objects on the stack that you want to keep.
3. Perform a System Halt ([ON][▲]).

The current directory is now HOME.

To Minimize Memory Usage for Array Calculations: Store arrays in variables and refer to them by name; avoid using them on the stack. Here's a comprehensive strategy for doing so.

1. Plan in advance how many variables you'll need, including intermediate results.
2. Create *small* arrays of the correct type (real or complex, vector or matrix), store them in variables, and then use RDM to adjust their sizes.
3. Perform calculations using the storage arithmetic commands in the STORE menu.
4. To act on individual elements, use GET, GETI, PUT, PUTI *with the variable's name*, or use algebraic syntax such as 'A(5,6)' EVAL and 'B(3)' STO; don't return the entire array to the stack.

21

Menus

Every operation, command, and function on the HP-28S is available on the keyboard or in a menu. When you select a menu, six *menu labels* appear in the bottom line of the display. These labels constitute one *menu row*, which indicates the current definitions of the six *menu keys* at the top of the keyboard. (The Cursor menu is an exception; its definitions are printed in white above the menu keys.)

In addition to the keys that select specific menus (such as ■ ARRAY or TRIG), the following keys control menu operations.

Key	Description
◂▸	**Cursor Menu On/Off.** When the Cursor menu is not active: selects the Cursor menu. When the Cursor menu is active: selects the previous menu.
■ CUSTOM	**Last Custom Menu.** Displays the Custom menu last created by the MENU command.
NEXT	**Next Menu Row.** Displays the next row of menu labels. If the last row is displayed, displays the first row.
■ PREV	**Previous Menu Row.** Displays the previous row of menu labels. If the first row is displayed, displays the last row.
■ MENUS	**Menu Lock.** Turns Menu Lock on and off. When Menu Lock is on, the shifted and unshifted "positions" are switched for the top three rows of the lefthand keyboard (letter keys A through R). When Menu Lock is on, pressing A selects the ARRAY menu and pressing ■ A writes the letter A.

Menus of Commands

The following menus contain keys for built-in operations, most of which are programmable commands. For a brief description of the commands in each menu, see appendix D, "Menu Map." The Reference Manual covers these menus in alphabetical order and describes them in detail.

The action of the keys in these menus depends on the entry mode, described on page 169.

Menu	Description
ALGEBRA	Algebra commands.
ARRAY	Vector and matrix commands.
BINARY	Integer arithmetic, base conversions, bit manipulations.
COMPLEX	Complex-number commands.
LIST	List commands.
LOGS	Logarithmic, exponential, hyperbolic functions.
MEMORY	User memory, directories.
MODE	Display, angle, recovery modes.
PLOT	Plotting commands.
PRINT	Printing commands.
PROGRAM BRANCH	Program branch structures.
PROGRAM CONTROL	Program control, halt, and single-step operations.
PROGRAM TEST	Flags, logical tests.
REAL	Real number commands.
SOLVE	Numerical and symbolic solution commands, the Solver.

Menu	Description
STACK	Stack manipulation.
STAT	Statistics and probability commands.
STORE	Storage arithmetic.
STRING	Character strings.
TRIG	Trigonometric functions, coordinate and angle conversions.

Menus of Operations

The following menus offer non-programmable operations.

Menu	Description
Cursor	For editing the command line. Described in chapter 18.
CATALOG	Catalog of commands, including USAGE submenu. Described in chapter 22.
UNITS	Units available for conversion. Described in chapter 14.

Menus of Variables

Menu	Description
Solver	Stores values and solves for variables in the current equation. Distinctive appearance (black letters against white menu label) indicates its distinctive action.
USER	Displays variables and subdirectories in current directory. The action of the keys depends on the entry mode, described on page 169.

Custom Menus

The command MENU can create a custom menu from a list of names and commands. The custom menu can be similar to the Solver menu or the USER menu.

- If the first element in the list is the command STO, followed by a sequence of names, MENU creates a *Custom Input* menu. This menu looks and acts like the Solver menu: pressing a menu key take a value from the stack and stores it in the corresponding variable. For an example, see chapter 27, "Interactive Programs."

- If the list contains a sequence of names and commands (the first element being different from STO), MENU creates a *Custom User* menu. This menu acts like a hybrid of the USER menu and a command menu. For an example, see "Changing Directories" on page 275.

22

Catalog of Commands

In chapter 1 you used the catalog of commands to check the correct spelling of a few commands and to check various combinations of arguments for the function +. This chapter reviews the operations available in the catalog, including the USAGE menu that shows correct combinations of arguments.

Pressing ■ CATALOG displays the command ABORT, which is the first command alphabetically, and the CATALOG menu.

Key	Description
NEXT	Advances the catalog to the next command. If you press and hold this key, the catalog advances repeatedly until you release the key.
PREV	Move the catalog back to the previous command. If you press and hold this key, the catalog moves back repeatedly until you release the key.
USE	Activates the USAGE menu display (see below) showing the stack arguments used by the command.
FETCH	Exits the catalog and writes the command's name in the command line.
QUIT	Exits the catalog, leaving the command line unchanged.

You can exit the catalog and clear any current command line by pressing ON.

Finding a Command

You can use the keys on the left-hand keyboard to move the catalog to a specific character.

- Pressing a letter key on the left-hand keyboard moves the catalog to the first command that starts with that letter. If there are no commands starting with that letter, the catalog moves to the last command starting with the previous letter.

- Pressing a non-letter character key (such as ▢ Σ) moves the catalog to the first command that starts with that character. If there are no commands starting with that character, the catalog moves to +, the first command that starts with a non-letter character.

- Pressing ▢ MENUS moves the catalog to →STR, the last entry in the catalog.

Checking Command Usage

You can check the correct stack argument types for the command currently displayed by the catalog. Pressing USE activates a second level of the catalog, called the USAGE menu, that shows all combinations of arguments for the command. If the command accepts more than one combination of arguments, the following menu keys appear. (If the command accepts only one combination of arguments, the labels NEXT and PREV don't appear.)

Key	Description
NEXT	Displays the next combination of arguments.
PREV	Displays the previous combination of arguments.
QUIT	Returns to the main catalog, with the current command displayed. You can then move through the catalog to other commands, or exit by pressing QUIT again.

You can exit both USAGE and the main catalog, and clear any current command line, by pressing ON .

23

Evaluation

All calculator operations, from simple keyboard calculations to complicated programs, involve evaluation. Some examples:

- When you key one or more objects into the command line and press ENTER, the command line is translated into a program, which is then evaluated.

- When you press a key on the USER menu in Immediate entry mode, the corresponding name is evaluated.

- When you perform step-by-step differentiation, you press EVAL to evaluate the expression in level 1.

- When you use the Solver to find numerical solutions, the procedure stored in the variable EQ is repeatedly evaluated.

It's easiest to understand calculator operations in terms of *delaying evaluation* and *causing evaluation*. Although the term "delaying evaluation" is new, the process is familiar: whenever you enter a quoted name or an algebraic, the object's delimiters indicate that you want to delay evaluation of the object—that you want the object to go on the stack.

Delayed evaluation is the basis for programming on any computing device, since otherwise a program would execute as soon as you wrote it. The HP-28S extends the concept in a uniform way to allow symbolic operations—you can use names and algebraics as data for symbolic calculations. For example, you choose when, if ever, you want to evaluate an expression. You can differentiate it, symbolically solve it, make substitutions for variables in it, and so on. Of course, you can also calculate its numerical value.

This chapter describes what happens when you evaluate the various types of objects. As a general introduction, consider the following *object classes*.

- *Data-class objects.* This class comprises real numbers, complex numbers, binary integers, strings, arrays, and lists. The "value" of a data object is exactly what it contains.

- *Name-class objects.* This class comprises global names and local names. The "value" of a name is generally the contents of a variable.

- *Procedure-class objects.* This class comprises algebraics and programs. The "value" of a procedure is the result of whatever process it defines.

In a rough way, these classes define what happens when you evaluate an object: it returns itself, or the contents of a variable, or the result of a process. It's not quite that simple, though, and more details are provided below for each object class.

Data-Class Objects

This is the simplest class of objects. Evaluating any data-class object returns the same object.

Note that lists are all-purpose data objects, since they can contain any object type. Consider a list of names: the names are protected from evaluation by the list, and they can't be evaluated until they're removed from the list.

Name-Class Objects

Generally, the "value" of a name is the contents of a variable. Evaluation of local names is simple and is described first, followed by evaluation of global names.

Evaluation of Local Names

As described in chapter 19, the use of local variables simplifies stack manipulations. The purpose of local variables is (1) to remove the variable's contents from the stack so it's out of the way and (2) to return a copy of the variable's contents whenever you need it. Consequently, evaluating a local name always returns the contents of the corresponding local variable to the stack.

Evaluation of Global Names

In general, evaluating a global name causes evaluation of the contents of the corresponding global variable. In other words, evaluating a global name has the same effect as evaluating the object it represents.

There are two exceptions to the general rule:

■ If no variable exists with the specified name, the name is returned to the stack. An undefined name used as a variable is called a *formal variable*.

■ If the contents of the specified variable is an algebraic, the algebraic *is not evaluated*. The calculator avoids evaluation of these objects so you can continue symbolic calculations. If you do want evaluation, execute the command EVAL with the algebraic in level 1. (To evaluate an algebraic repeatedly until it produces a numerical result, execute →NUM.)

If the variable contains a data-class object, evaluating the variable's name is equivalent to simply recalling the variable's contents. However, evaluating a variable's name can lead to a long chain of evaluations. For example, if a variable contains a name, and that name is the name of a second variable, and the second variable contains a name, and that name is the name of a third variable, then evaluating the name of the first variable ultimately causes evaluation of contents of the third variable.

> **Note**
>
> Do not create a variable whose value is its own name, such as a variable named X that contains the name 'X'. Evaluating such a variable causes an endless loop. To halt an endless loop, you must perform a system halt (ON ▲), which also clears the stack.

Similarly, do not create variables that reference one another in a circular definition. Evaluating a variable included in a circular definition also causes an endless loop.

Procedure-Class Objects

Generally, the "value" of a procedure is the result of whatever process it defines. Programs are the most general procedure-class objects, so they're described first, followed by algebraics.

Evaluation of Programs

A program is a sequence of objects and commands. This manual uses the terms "evaluate a program" and "execute a program" interchangably. In general, evaluating a program takes the program's contents in order, putting each object on the stack and executing each command. There are two additional points to remember:

- Unquoted names are evaluated, while quoted names go on the stack. Names are quoted expressly to delay evaluation, as discussed on page 57.

- Program structures are executed according to their own rules. In part 1 you wrote several user functions, which contain a *local-variable structure*. Program structures are described in chapter 26.

The rules for evaluating names and evaluating programs lead to one of the fundamental ideas in programming the HP-28S. For this discussion, "program" means a program stored in a variable, and "name of a program" means the name of the variable that contains a program.

The fundamental idea is called *structured programming*. It means that a complicated task is broken into subtasks, and a program is written for each subtask. The main program can now be relatively simple, reflecting the overall logic of the task. It can execute each subtask simply by including the unquoted name of the program for that subtask. If a subtask is executed more than once, the unquoted name can be included more than once. If other main programs use the same subtask, they can execute the subtask in the same way.

Structured programming is demonstrated in "Expanding and Collecting Completely" on page 253, "Displaying a Binary Integer" on page 257, and "Median of Statistics Data" on page 270.

Evaluation of Algebraics

Each algebraic is equivalent to a program that contains only unquoted names and functions. Evaluating an algebraic produces the same result as evaluating the corresponding program: unquoted names are evaluated, and functions are executed. This topic is also discussed in "Evaluation of Algebraic Objects" in the Reference Manual.

The result of evaluating a name depends on the existence of a variable with that name, as described in "Evaluation of Global Names" above. Some examples:

- If a name refers to a user function, you can use the user function's name like a built-in function. Evaluation of the algebraic causes execution of the user function. The arguments to the user function, enclosed in parentheses and following the user function's name, are part of the algebraic.

- If a name refers to a program that takes no arguments from the stack and returns exactly one result, you can use the program's name to refer (indirectly) to the result. Evaluation of the algebraic causes execution of the program, so in effect the program's name is replaced by the result. For examples, see "Summary Statistics" on page 262.

- If a name refers to a second algebraic, evaluation of the first algebraic *doesn't* cause evaluation of the second algebraic. Instead, the second algebraic effectively replaces its name in the first algebraic.

A special case among functions is the function " = ", which distinguishes equations from expressions. Depending on the Result mode (Symbolic or Numerical), executing = returns an equation or a numerical result.

- In Symbolic Result mode, evaluating an equation produces a new equation. The new left-hand expression is the result of evaluating the original left-hand expression. The new right-hand expression is the result of evaluating the original right-hand expression.

- In Numerical Result mode, evaluating an equation produces the numerical difference between the original left-hand expression (numerically evaluated) and the original right-hand expression (numerically evaluated).

The next section describes Result modes in more detail.

Evaluation of Functions

When a function is evaluated, its action depends on the current Result mode, which can be Symbolic or Numerical. These modes are also described in the next chapter, "Modes."

Symbolic Result Mode. This is the default case, where a function accepts symbolic arguments and returns symbolic results. The action of functions in Symbolic Result mode is evident when you calculate with names and expressions to create larger expressions.

Numerical Result Mode. This alternative is used in plotting and by the Solver. Its purpose is to ensure a numerical result from the function. In this mode, functions repeatedly evaluate symbolic arguments, accepting only numerical arguments and returning numerical results.

You can force evaluation of an object until it returns a numerical result by executing →NUM (*to number*); in chapter 5 you did this to return a numerical value for π.

Note In Numerical Result mode, do not evaluate a variable whose value includes its own name, such as a variable named X that contains the expression 'X+Y'. Evaluating such a variable causes an endless loop. To halt an endless loop, you must perform a system halt (ON ▲), which also clears the stack.

Similarly, do not create variables that reference one another in a circular definition. Evaluating a variable included in a circular definition also causes an endless loop.

24

Modes

You can affect the results of many operations by selecting a mode. Some modes, such as angle mode (Degrees or Radians), can be selected by pressing a menu key. The mode's menu label includes a small square when the mode is selected. For example, the menu label for Radians angle mode appears as `RAD■` when that mode is selected.

Most modes, such as Beeper mode (on or off), can be selected by setting or clearing a user flag, using the commands `SF` (*set flag*) and `CF` (*clear flag*). For example, flag 51 controls Beeper mode, so you can turn the beeper off by executing `51 SF`.

This chapter describes how the modes affect calculator operation and lists the associated menu labels and flags. Also shown are annunciators that appear when a mode is selected. For each mode, the selection listed first is the default selection, active following Memory Lost.

General Modes

These modes affect computations and the beeper.

Angle Mode

This mode determines whether real numbers represent angular measure in degrees or in radians. This affects arguments to trigonometric functions and the results from inverse trigonometric functions.

Degrees Mode (`DEG▪` **, Flag 60 Clear).** Real numbers represent angular measure in degrees.

Radians Mode (`RAD▪` **, Flag 60 Set,** (2π) **).** Real numbers represent angular measure in radians.

Beeper Mode

This mode controls whether the calculator makes sounds when an error occurs or BEEP is executed.

Beeper On (Flag 51 Clear). The calculator makes sounds.

Beeper Off (Flag 51 Set). The calculator is silent.

Principal Value

A solution returned by ISOL or QUAD generally requires arbitrary signs ($+1$ or -1) and integers (0, 1, 2, ...) to represent all possible solutions. This mode determines whether arbitrary signs and integers are included in solutions generated by ISOL or QUAD.

Principal Value Off (Flag 34 Clear). Solutions returned by ISOL and QUAD include variables s1, s2, ... , for arbitrary signs and n1, n2, ... , for arbitrary integers.

Principal Value On (Flag 34 Set). ISOL and QUAD take arbitrary signs to be $+1$ and arbitrary integers to be 0.

Constants Mode

This mode affects whether evaluation of a symbolic constant (e, i, MINR, MAXR, or π) returns its numerical value. In Numerical Results mode (flag 36 clear), evaluation of a symbolic constant returns its numerical value regardless of Constants Mode.

Symbolic Constants (Flag 35 Set). Evaluation of a symbolic constant returns its symbolic form.

Numerical Constants (Flag 35 Clear). Evaluation of a symbolic constant returns its numerical value.

Results Mode

The current Result mode affects the result of evaluating a function when its arguments are symbolic.

Symbolic Results (Flag 36 Set). Given symbolic arguments, functions return symbolic results.

Numerical Results (Flag 36 Clear). Functions always return numerical results. To do so, functions evaluate symbolic arguments repeatedly to determine their numerical values. Evaluation of a symbolic constant returns its numerical value regardless of Constants Mode.

Entry and Display Modes

These modes affect how objects are entered and displayed.

Entry Mode

The current entry mode affects the result when you press a command, function, or User menu key. The entry mode automatically changes when you press ⏍, ▮⏍, or ⏍; you can also change it manually by pressing ⍺. The appearance of the cursor indicates the current entry mode. For details, see chapter 18, "The Command Line."'

Immediate Entry (Open Cursor). The command line is executed when you press a command, function, or User menu key.

Algebraic Entry (Partly Filled Cursor). The command line is executed when you press a command key.

Alpha Entry (Solid Cursor, α). The command line is executed only when you press ENTER.

Replace or Insert Mode

Pressing ⌈INS⌉ in the cursor menu switches between Replace and Insert modes. The appearance of the cursor indicates Replace or Insert mode.

Replace Mode (Box Cursor). New characters replace existing characters.

Insert Mode (Arrow Cursor). New characters are inserted between existing characters.

Uppercase or Lowercase

Pressing ⌈LC⌉ switches between Uppercase and Lowercase modes.

Uppercase Mode. Pressing a letter key writes an uppercase letter in the command line.

Lowercase Mode. Pressing a letter key writes a lowercase letter in the command line.

Level 1 Display

Many objects are too large to show on a single display line. You can choose to use more than one line to display the object in level 1, if needed, or to use only one line regardless of the object's size. This choice affects the printed output in Trace mode.

ML On (ML∎ , Flag 45 Set). Objects in level 1 are displayed on more than one line if needed.

ML Off (ML , Flag 45 Clear). Objects in level 1 are displayed on only one line.

Decimal Point Mode

The comma and the period share the roles of radix mark (to distinguish the integer part of a number from the fractional part) and separator (to distinguish objects in the command line; the space is always a separator). You can assign these roles to the comma and period in either order.

RDX, Off (`RDX,` **, Flag 48 Clear).** The period is the radix mark (decimal point), and the comma is a separator.

RDX, On (`RDX, ▪` **, Flag 48 Set).**
The comma is the radix mark, and the period is a separator.

Number Format

These modes determine the number of decimal places displayed for real numbers. The commands FIX, SCI, and ENG require a real-number argument n. The current number format mode also affects the command RND (*round*).

STD Format (`STD▪` **).** Real numbers are displayed with a decimal point or an exponent only if necessary.

FIX Format (`FIX▪` **).** Real numbers are displayed with n decimal places. An exponent is displayed only if necessary.

SCI Format (`SCI▪` **).** Real numbers are displayed as a mantissa, which is less than 10 and contains n decimal places, and an exponent.

ENG Format (`ENG▪` **).** Real numbers are displayed as a mantissa, which contains $n + 1$ digits, and an exponent that is a multiple of 3.

Integer Base

You can choose the base used for entering and displaying binary integers. The choice of base doesn't affect the internal structure of binary integers, which are always treated as a sequence of bits.

DEC Base (⬛DEC▪). Binary integers entered without base markers are interpreted in base 10. All binary integers are displayed in base 10 and show a "d" base marker.

HEX Base (⬛HEX▪). Binary integers entered without base markers are interpreted in base 16. All binary integers are displayed in base 16 and show an "h" base marker.

OCT Base (⬛OCT▪). Binary integers entered without base markers are interpreted in base 8. All binary integers are displayed in base 8 and show a "o" base marker.

BIN Base (⬛BIN▪). Binary integers entered without base markers are interpreted in base 2. All binary integers are displayed in base 2 and show a "b" base marker.

Binary Integer Wordsize

The current wordsize can range from 1 bit through 64 bits. It controls how binary integers are displayed; also, binary integers are truncated to the current wordsize when used as arguments or returned as results. To set the wordsize to n, execute n STWS (*store wordsize*).

Recovery Modes

The recovery modes determine whether copies are made of command lines, of the stack, and of arguments to commands. These copies can help you to recover if you make a mistake.

CMD Mode

This mode determines whether a copy of the command line is saved when you press ENTER (or a key that performs ENTER).

CMD On (⬛CMD▪). Command lines are saved for recovery by
⬛ COMMAND .

CMD Off (`CMD` **).** Command lines are not saved.

UNDO Mode

This mode determines whether a copy of the stack is saved when you press ENTER (or a key that performs ENTER).

UNDO On (`UNDO▪` **).** The stack is saved for recovery by ▇ UNDO .

UNDO Off (`UNDO` **).** The stack is not saved.

LAST Mode

This mode determines whether copies of arguments are saved when a command is executed.

LAST On (Flag 31 Set, `LAST▪` **).** Arguments are saved for recovery by LAST or in case of error.

LAST Off (Flag 31 Clear, `LAST` **).** Arguments are not saved. If an error occurs, the arguments to the last command are not returned to the stack.

Mathematical Exceptions

Certain errors that can arise during ordinary real number calculations are called *mathematical exceptions*. An exception can act as an ordinary error and halt the calculation, or it can supply a default result and allow the calculation to proceed.

Infinite Result Action

An Infinite Result exception occurs when a calculation returns an infinite result. Examples include evaluation of 'LN(0)', 'TAN(90)' (in Degrees angle mode), or 'X/0'.

Infinite Result Error On (Flag 59 Set). Infinite Result exceptions are errors.

Infinite Result Error Off (Flag 59 Clear). Infinite Result exceptions return ±9.99999999999E499 and set the Infinite Result indicator (flag 64).

Overflow Action

An Overflow exception occurs when a calculation would return a finite result whose absolute value is greater than the largest machine-representable number. Examples include the evaluation of `'9E499+9E499'`, `'EXP(5000)'`, or `'FACT(2000)'`.

Overflow Error Off (Flag 58 Clear). Overflow exceptions return ±9.99999999999E499 and set the Overflow indicator (flag 63).

Overflow Error On (Flag 58 Set). Overflow exceptions are errors.

Underflow Action

Underflow exceptions occur when a calculation returns a finite result whose absolute value is smaller than the smallest machine-representable number. Examples include the evaluation of `'1E-499/2'` or `'EXP(-5000)'`.

Underflow Error Off (Flag 57 Clear). Underflow exceptions return the default result 0. They set the Underflow+ indicator (flag 62) or the Underflow− indicator (flag 61), depending on the sign of the actual result.

Underflow Error On (Flag 57 Set). Underflow exceptions are errors. They return the error message `Negative Underflow` or `Positive Underflow`, depending on the sign of the actual result.

Printing Modes

The following modes give you greater flexibility in printing.

Trace Printing

You can automatically print a record of your calculations.

Trace Printing Off (TRAC , Flag 32 Clear). No automatic printing occurs.

Trace Printing On (TRAC■ , Flag 32 Set). Each time the command line is executed, the calculator prints the contents of the command line, the operation that caused execution, and the result in level 1.

Auto CR Mode

Generally you want to send data to the printer and print the data with a single command. In other cases, such as printing graphics, you want to accumulate data in the printer without printing. This mode determines whether print commands automatically cause printing.

Auto CR (Flag 33 Clear). Print commands send Carriage Right at the end of transmission, causing the data to be printed.

Accumulate Print Data (Flag 33 Set). Print commands send data without Carriage Right, causing the data to accumulate in the printer buffer.

Print Speed

The calculator can't sense when the printer is ready for more data, so it computes the rate at which data can safely be transmitted. This mode determines whether the computation is made for a printer powered by batteries or one that is powered by an adaptor.

Normal Print Speed (Flag 52 Clear). The calculator sends data at a rate suitable for battery-powered printing.

Faster Print Speed (Flag 52 Set). The calculator sends data at a rate suitable for adaptor-powered printing.

Print Spacing

This mode determines whether blank lines are automatically printed.

Single-Space Printing (Flag 47 Clear). No blank lines are printed automatically.

Double-Space Printing (Flag 47 Set). One blank line is automatically printed between every two text lines.

25

System Operations

This chapter describes special key combinations that interrupt normal HP-28S operation. These system operations include printing the display, adjusting display contrast, halting programs, resetting memory, and performing diagnostic tests.

All system operations begin by pressing the $\boxed{\text{ON}}$ key. You can cancel any system operation by pressing $\boxed{\text{DEL}}$ before you release $\boxed{\text{ON}}$.

The table below shows the keystrokes for system operations, followed by a description of each.

System Operations

Name	Keystrokes
Print Display	$\boxed{\text{ON}}\boxed{\text{L}}$
Contrast Control	$\boxed{\text{ON}}\boxed{+}$ or $\boxed{\text{ON}}\boxed{-}$
Attention	$\boxed{\text{ON}}$
System Halt	$\boxed{\text{ON}}\boxed{\blacktriangle}$
Memory Reset	$\boxed{\text{ON}}\boxed{\text{INS}}\boxed{\blacktriangleright}$
Repeating Test	$\boxed{\text{ON}}\boxed{\blacktriangleleft}$
Keyboard Test	$\boxed{\text{ON}}\boxed{\text{NEXT}}$
Cancel System Operation	$\boxed{\text{ON}}\boxed{\text{DEL}}$

Printing the Display

To print the current display image:

1. Press and hold [ON].

2. Press [L] (the key with "PRINT" above it).

3. Release [ON].

Contrast Control

To change the display contrast:

1. Press and hold [ON].

2. Press [+] to increase the contrast or press [−] to decrease the contrast. As long as you hold down [ON], you can press [+] or [−] repeatedly or continuously to find the best contrast.

3. Release [ON].

Clearing Operations

There are three clearing operations, given below in order of increasing severity.

Attention

To return to the normal stack display, execute Attention by pressing [ON]. In some cases you may need to press [ON] twice. Attention has following effects:

■ Clears the command line.

■ Cancels all command or procedure execution.

■ Exits special operations such as FORM, PLOT, and catalogs.

■ Restarts normal keyboard operation.

System Halt

To halt a program that doesn't respond to [ON], execute System Halt as follows:

1. Press and hold [ON].
2. Press [▲].
3. Release [ON].

System Halt has the following effects:

- All the effects of Attention.
- Clears all suspended programs and local variables.
- Clears the stack.
- Clears items saved for recovery (CMD, UNDO, LAST).
- Clears the custom menu.
- Selects HOME as the current directory.
- Activates the cursor menu.
- Selects Trace Printing Off mode.

Memory Reset

To reset all memory:

1. Press and hold [ON].
2. Press and hold [INS] and [▶].
3. Release [INS] and [▶].
4. Release [ON].

Memory Reset has the following effects:

- All the effects of Attention and System Halt.
- Purges all user variables and directories.
- Resets all user flags to their default values.
- Beeps and displays Memory Lost in display line 1.

Test Operations

There are two system operations for testing the calculator. The first is a repeating test of the electronics, which runs without assistance. The second is a keyboard test, which requires you to press all the keys in a specified sequence. Both tests perform a System Halt.

Repeating Test

To perform the repeating test:

1. Start the test.

   a. Press and hold [ON].

   b. Press [◄].

   c. Release [ON].

2. The display shows horizontal and vertical lines, a blank display, a random pattern, and then it briefly displays the result of the test before starting over.

   ■ The message OK-28S indicates that the calculator passed the test.

   ■ A message such as 1 FAIL indicates that the calculator failed the test. The number indicates the nature of the failure.

     If you interrupt the test by pressing a key, the test returns a failure message because it didn't expect any keystrokes. *Such a failure message doesn't indicate a problem with the calculator.*

3. Exit the test by performing a System Halt.

   a. Press and hold [ON].

   b. Press [▲].

   c. Release [ON].

Keyboard Test

To perform the keyboard test:

1. Start the test.

    a. Press and hold $\boxed{\text{ON}}$.

    b. Press $\boxed{\text{NEXT}}$.

    c. Release $\boxed{\text{ON}}$.

2. The calculator displays KEYBOARD TEST.

    a. Test the first row of the lefthand keyboard by pressing $\boxed{\text{A}}$ $\boxed{\text{B}}$ $\boxed{\text{C}}$ $\boxed{\text{D}}$ $\boxed{\text{E}}$ $\boxed{\text{F}}$.

    b. Test the second through sixth rows of the lefthand keyboard in the same way.

    c. Test the first row of the righthand keyboard by pressing $\boxed{\text{INS}}$ $\boxed{\text{DEL}}$ $\boxed{\blacktriangle}$ $\boxed{\blacktriangledown}$ $\boxed{\blacktriangleleft}$ $\boxed{\blacktriangleright}$.

    d. Test the second through seventh rows of the righthand keyboard in the same way. (Press the $\boxed{\text{ON}}$ key in the correct order—it won't interrupt the test.)

3. If you've pressed the keys in the correct order and the keyboard is working properly, the calculator displays OK-28S. A message such as 1 FAIL indicates that you didn't follow the correct order or the calculator failed the test. The number indicates the nature of the failure.

4. Press $\boxed{\text{ON}}$.

Part 3

Programming

Program Structures

Many programs are equivalent to a series of immediate-execute key-board computations. Objects go on the stack and commands are executed, producing the desired result. These programs are simply a record of the objects and commands, written in the same order as you would execute them from the keyboard. However, there are features you can use in programs that go beyond simple keystrokes.

For example, in part 1 you wrote programs that created local vari-ables. The special command ⇥, followed by one or more names, followed by a procedure, is called a *local-variable structure*. You can't execute the command ⇥ from the keyboard; it must appear in the same program as the names and procedure that constitute the entire program structure.

This chapter first reviews the local-variable structure. It then describes additional program structures that conduct tests and modify program execution based on the result. All commands for these program struc-tures appear in the PROGRAM BRANCH menu. Be sure to read the first example in "Conditional Structures" on page 223, which intro-duces concepts used in the remainder of the chapter.

Local-Variable Structure

In part 1 you wrote several user functions, which are the most impor-tant application of the local-variable structure. There are two requirements for user functions. They must:

- Explicitly indicate their arguments.
- Return exactly one result.

For example, the user function COT (from chapter 5) was written:

$$\ll \rightarrow \text{x } \text{'INV(TAN(x))' } \gg$$

Here the local-variable structure stores one argument in a local variable x (satisfying the first requirement) and evaluates the expression 'INV(TAN(x))' (satisfying the second requirement). The user function O→G (from chapter 14) included a program rather than an expression but, since the program returned exactly one result, O→G also satisfied the second requirement.

These requirements apply only to user functions. More generally, local variables are used as a substitute for stack manipulations. The following example returns the sum and difference of two numbers. Since it returns two results, it can't be a user function.

$$\ll \rightarrow \text{x y } \ll \text{x y } + \text{x y } - \gg \gg$$

For more examples, see the programs in chapter 28. They use local-variable structures more often to avoid stack manipulations than to create user functions.

Conditional Structures

Conditional structures enable a program to test a specified condition and make a decision based on the result of the test. This section first gives an example of a conditional structure. It uses that example to discuss program structures in general, and then it describes other types of conditional structures.

Suppose you're writing a program that uses the variable x, and you want to calculate $(\sin x)/x$. A problem arises because the quotient is undefined when $x = 0$. The following example returns $(\sin x)/x$ if $x \neq 0$, or returns 1 if $x = 0$.

```
IF X 0 ≠ THEN X SIN X / ELSE 1 END
```

Here's how this structure works when you execute the program:

1. The IF command simply marks the start of the structure. It can be anywhere before the THEN command.

2. X is evaluated.

3. The number 0 goes on the stack.

4. The command ≠ takes the value of X and the number 0 as arguments.

   ■ If the arguments are "not equal", ≠ returns 1.

   ■ If the arguments are *not* "not equal", ≠ returns 0.

5. The command THEN takes 1 or 0 as its argument.

   ■ If its argument is 1, THEN evaluates the program up to ELSE (namely X SIN X ⁄).

   ■ If its argument is 0, THEN evaluates the program from ELSE to END (namely 1).

6. Program execution continues after the END command.

Before continuing with specifics about conditional structures, here's some general information about program structures.

Program-Structure Commands. The commands IF, THEN, ELSE, and END are examples of *program-structure commands*. The order and meaning of these commands are similar to their use in English. You can't use program-structure commands as flexibly as other commands; they work only in the combinations described in this chapter.

Test Functions and Commands. The function ≠ is called a *test function*. Given two numbers, ≠ returns 1 or 0, indicating whether the test is true or false. Other test functions are ⟨, ≤, ≥,⟩, and ==. (Remember that = is used for equations, not to test equality.) Given symbolic arguments, test functions return a symbolic result.

There are also test commands that always return 1 or 0. For example, the test command SAME is similar to ==, but it simply tests whether the two objects are identical. Additional test commands are available for working with flags (described next). For more information about test functions and commands, see "PROGRAM TEST" in the Reference Manual.

Flags. The numbers 1 and 0 that are returned by test commands are called *stack flags*. Because they represent the truth or falsity of the test, 1 is called a *true flag*, and 0 is called a *false flag*.

The term "flag" also refers to the built-in *user flags*. They are numbered 1 through 64; flags 31 through 64 have specific meanings to the calculator, while flags 1 through 30 can represent any true/false distinction you wish. You can effectively store a stack flag in a user flag, since both represent a truth value. For example, the sequence

```
IF A B < THEN 12 SF ELSE 12 CF END
```

sets flag 12 if A < B, or it clears flag 12 if A ≥ B. You can later test whether flag 12 is set by the sequence

```
IF 12 FS? THEN ...
```

which returns the same truth value as the original test A B <. The advantage to this technique is that the truth value of the original test is preserved, even if the values of A and B have changed. The commands for changing and testing user flags appear in "PROGRAM TEST" in the Reference Manual. For the remainder of this chapter, "flag" refers to a stack flag.

Clauses. The objects and commands between two program-structure commands are called a *clause*. Each clause is handled as a single entity by the program structure. A clause is labeled by its logical role or by the command that precedes it. In the first example:

- The clause between IF and THEN (X 0 ≠) is called the *test clause* or *IF clause*.
- The clause between THEN and ELSE (X SIN X /) is called the *true clause* or *THEN clause*.
- The clause between ELSE and END (1) is called the *false clause* or *ELSE clause*.

The clauses in the example represent simple numerical calculations, but you can include any sequence of objects and commands. In effect, a clause is like a subprogram within the program. If you write a separate program that contains the clause and store this program in a variable, you can use the variable's name as the entire clause. In this case a simple-looking structure like

```
IF A THEN B ELSE C END
```

can represent a complicated decision process with two possible complicated results, depending on the contents of A, B, and C.

IF ... THEN ... ELSE ... END

Using the terminology just defined, the evaluation of this conditional structure can be described as follows: The IF clause is evaluated and returns a flag. If the flag is true, the THEN clause is evaluated; if the flag is false, the ELSE clause is evaluated.

For another example of this structure, see "FIB2 (Fibonacci Numbers, Loop Version)" on page 248.

IFTE (If-Then-Else-End Function)

The first example in this chapter can be written in algebraic syntax by using the function IFTE:

```
'IFTE(X≠0,SIN(X)/X,1)'
```

This form is handy for symbolic calculations. If you execute the program-structure version while X is undefined, this algebraic form is the result. The arguments to IFTE must be representable in algebraic syntax; to include RPN commands in the conditional, you must use the program-structure form.

The IFTE function is used in "FIB1 (Fibonacci Numbers, Recursive Version)" on page 247.

IF ... THEN ... END

If an ELSE clause isn't required—that is, if the choices are to do something or do nothing—you can omit ELSE from the program structure. The following example ensures that the object in level 1 is greater than the object in level 2 by swapping them if necessary.

```
IF DUP2 ≤ THEN SWAP END
```

Note the use of DUP2 to make copies of the objects. The copies are then consumed by the comparison ≤. For another example of IF ... THEN ... END, see "SORT (Sort a List)" on page 270.

IFT (If-Then-End Command)

You could write the previous example by using the command IFT instead of the program structure:

```
DUP2 ≤ « SWAP » IFT
```

The sequence DUP2 ≤ leaves a flag on the stack, the program « SWAP » goes on the stack, and the command IFT takes the flag and the program as arguments. If the flag is true, IFT evaluates the program; if the flag is false, IFT drops the program. The result is identical to the program-structure form.

Error Traps

In some cases you can predict that an error might occur during program execution. Normally an error cancels program execution; but if you *trap* the offending command by enclosing it in a special program structure, the program can continue execution when the error occurs.

Remember the problem with $(\sin x)/x$—it causes an Infinite Result error when $x = 0$. Another method for defining $(\sin 0)/0 = 1$ would be:

```
IFERR X SIN X / THEN DROP2 1 END
```

This means: Evaluate the IFERR clause (`X SIN X /`). If an error occurs, evaluate the THEN clause (`DROP2 1`).

This example includes the command DROP2 to drop the two zeros that caused the error. Note that this assumes that LAST is enabled. If LAST is disabled, the zeros aren't present and the DROP2 command is inappropriate. Be sure to consider the state of LAST when using error traps.

Another example of `IFERR ... THEN ... END` appears in "BDISP (Binary Display)" on page 259. Also, you can include an ELSE clause to be evaluated only if an error doesn't occur, using the form

<div align="center">

`IFERR ... THEN ... ELSE ... END`

</div>

Definite Loop Structures

Loop structures contain a *loop clause* that is repeatedly evaluated. In a *definite loop structure*, the program specifies in advance how many times to evaluate the loop clause. Another type of program structure, called an *indefinite loop structure*, uses a test clause to determine whether to repeat evaluation of the loop clause. This section describes definite loop structures; indefinite loop structures are described on page 231.

START... NEXT

The following example sounds a tone four times.

<div align="center">

`1 4 START 440 .1 BEEP NEXT`

</div>

This structure works as follows:

1. The command START takes the values 1 and 4 from the stack and creates a *counter*. The counter will be used to keep track of how many times to repeat the loop. The value 1 specifies the *initial value* of the counter, and the value 4 specifies its *final value*.

2. The loop clause `440 .1 BEEP` is executed.

3. The command NEXT adds 1 to the counter.

4. The current counter value is compared with the final counter value.

- If the current counter value doesn't exceed the final counter value, steps (2), (3), and (4) are repeated.

- If the current counter value exceeds the final counter value, the definite loop structure is completed, and program execution continues after the NEXT command.

In this example, steps (2), (3), and (4) are repeated four times. The loop counter is first incremented from 1 to 2, then to 3, then to 4, and then to 5. At this point it exceeds the final value 4, so the definite loop structure ends. Note that step (1) is performed before any tests are made, so the loop clause is always evaluated at least once. For another example of START ... NEXT, see "FIB2 (Fibonacci Numbers, Loop Version)" on page 248.

FOR *counter*... NEXT

In many cases it's handy to use the current value of the counter as a variable in the loop clause. To do so, replace START by FOR *name*. The counter becomes a local variable with the specified name. As before, this manual follows the convention of writing local names in lowercase letters to help you distinguish them from global names. The following example puts the first five square integers on the stack.

```
1 5 FOR x x SQ NEXT
```

The sequence FOR x is executed only once. The sequence x SQ is the loop clause, which is executed repeatedly.

The examples so far have specified an initial counter value of 1, but any integer is acceptable. Since you're using the counter as a variable, set the initial and final counter values to the desired initial and final variable values. The following example puts the third through ninth square integers on the stack.

```
3 9 FOR x x SQ NEXT
```

For another example, see "BDISP (Binary Display)" on page 259.

... *increment* STEP

The command NEXT always increments the counter by 1. To specify a different increment, replace NEXT by *n* STEP, where *n* is the desired increment. STEP is commonly used following FOR *counter*, as demonstrated in the examples below, but it can also be used following START. The following example puts the *odd* square integers from 1^2 through 5^2 on the stack.

```
1 5 FOR x x SQ 2 STEP
```

The loop clause x SQ 2 is executed three times. The command STEP first increments the counter from 1 to 3, then to 5, and then to 7. At this point the current value of the counter exceeds the final value 5, so the definite loop structure ends.

The examples so far have used *ascending* values of the counter. For *descending* values of the counter, you can specify a negative increment. The following example puts the odd square integers from 5^2 through 1^2 on the stack.

```
5 1 FOR x x SQ -2 STEP
```

The sequence -2 STEP decrements the counter from 5 to 3, then 1, and then -1. At this point the current value of the counter is less than the final value 1, so the definite loop structure ends.

The program "SORT (Sort a List)" on page 270 uses -1 STEP to decrement the counter by one. In this case STEP alters the value of the counter by 1, as does NEXT, but the counter decreases rather than increases.

Indefinite Loop Structures

If you can't specify in advance how many times to repeat a loop, you can write an *indefinite loop structure* that contains both a loop clause and a test clause. The clauses are executed alternatingly, with the result of the test clause determining whether to continue.

This section describes two types of indefinite loop structure. The first, DO... UNTIL... END, executes the loop clause before the test clause. Consequently, the loop clause is always executed at least once. The second type, WHILE... REPEAT... END, executes the test clause first. Consequently, in some cases the loop clause is never executed.

DO...UNTIL...END

The following example evaluates an object repeatedly until evaluation doesn't change the object. Since evaluation must occur at least once before the first test can be made, this example uses DO... UNTIL... END.

        DO DUP EVAL UNTIL DUP ROT SAME END

This structure works as follows:

1. The loop clause DUP EVAL is executed, leaving the object and the evaluated result on the stack.

2. The test clause DUP ROT SAME is evaluated, leaving the evaluated result and a flag on the stack. The flag indicates whether the object and the evaluated result are the same.

3. The flag is taken from the stack. Its value determines whether the loop structure is repeated.

   ■ If the flag is false, steps (1), (2), and (3) are repeated.

   ■ If the flag is true, the loop structure ends.

Suppose that you want to completely evaluate 'A+B', where A contains 'P+Q' and P contains 2. The first evaluation of the loop clause returns 'A+B' and 'P+Q+B'. These expressions are not the same, so the loop clause is evaluated a second time, returning 'P+Q+B' and '2+Q+B'. These expressions are not the same, so the loop clause is evaluated a third time, returning '2+Q+B' and '2+Q+B'. These expressions are the same, so the loop structure ends.

The effect of this example is similar to the effect of →NUM, except →NUM causes an error if a name is undefined. For a more versatile version of this example, see "MULTI (Multiple Execution)" on page 253.

WHILE ... REPEAT ... END

The following example takes any number of vectors from the stack and adds them to the current statistics matrix. Since it needs to test whether the object in level 1 is a vector *before* attempting to add it, this example uses WHILE ... REPEAT ... END.

```
WHILE DUP TYPE 3 == REPEAT Σ+ END
```

This structure works as follows:

1. The test clause DUP TYPE 3 == is evaluated, leaving a flag on the stack. The flag indicates whether the object in level 2 is a real vector.

2. The flag is taken from the stack. Its value determines whether the loop clause is executed.

- ■ If the flag is true, the loop clause Σ+ is executed, adding the vector to the current statistics matrix, and steps (1) and (2) are repeated.

- ■ If the flag is false, the loop structure ends.

Note that WHILE ... REPEAT ... END ends when the flag is false, but DO ... UNTIL ... END ends when the flag is true. If you need to change the truth value of a test clause, add NOT as the last command: WHILE ... NOT REPEAT or UNTIL ... NOT END.

For another example of WHILE ... REPEAT ... END, see "PAD (Pad With Leading Spaces)" on page 257.

Nested Program Structures

Since a clause in a program structure is like a subprogram, the clause itself can contain a program structure. The structure inside the clause is called the *inner* structure, and the structure that contains the clause is called the *outer* structure. The program "SORT (Sort a List)" on page 270 demonstrates nested definite loops.

There is no limit to the levels of nesting, except perhaps your ability to understand the logic. In some cases it's easier to store the inner structures in programs and use their names as clauses in the outer structures.

27

Interactive Programs

Some programs require direction from the user—that is, from you when you're running the program. When the user must supply values for variables, a program can ask for input. When the user must choose among several alternatives, a program can ask for a choice.

This chapter demonstrates how a program can ask for input or a choice, using the following commands from the PROGRAM CONTROL menu.

Command	Description
HALT	Suspend program execution.
s WAIT	Suspend program execution for s seconds.
KEY	Return a key string if a key was pressed.
f s BEEP	Sound a tone of frequency f for s seconds.
CLLCD	Clear the display.
n DISP	Display an object in line n of the display.
CLMF	Restore the normal display when the program completes execution.

Asking for Input

The following sequence creates a custom input menu for variables A, B, and C, sounds a tone to alert the user, and halts for input.

```
... { STO A B C } MENU 440 .1 BEEP HALT ...
```

The displayed menu shows the labels ⌐ A ⌐, ⌐ B ⌐, and ⌐ C ⌐, which resemble labels in the Solver menu. After entering a value on the stack, the user can simply press one of these keys to store the value in the corresponding variable. After entering the values, the user must press ■ CONT to continue program execution.

Asking for a Choice

For complex tasks it's best to write a series of small programs, each performing a small task. In some cases the user has several options for performing one of the tasks. One approach is to write alternative programs to perform that task.

Assume that one task is completed, and the user must choose among the programs HOP, SKIP, and JUMP for the next task. The following sequence creates a custom user menu for programs HOP, SKIP, and JUMP, sounds a tone to alert the user, and ends program execution.

```
... ( HOP SKIP JUMP ) MENU 440 .1 BEEP »
```

The displayed menu shows the labels HOP, SKIP, and JUMP, which resemble labels in the User menu. When the user presses one of the menu keys, the next task is performed. That task may end with a similar sequence, offering the user a different set of options; and so on throughout the entire complex task.

A More Complicated Example

The example below displays a message, waits until the user presses a key, and checks that the key is defined (that is, represents one of the choices). If the key is defined, the corresponding action is performed; if the key isn't defined, an error message is displayed and the process starts over.

This example uses program structures described in the previous chapter. There is an "outer" DO ... UNTIL ... END structure that repeats until the user presses a defined key. The outer DO clause contains an "inner" DO ... UNTIL ... END structure that repeats until the user presses a key. The outer UNTIL clause contains a conditional that

displays an error message if the key is undefined. In the listing below, the indentation marks the outer structure, the clauses, the inner structures, and their clauses.

Sequence	Comments
{ "Apple" "Banana" "Cherry" }	This list contains the possible outcomes. It remains on the stack until the following DO...UNTIL...END structure returns 1, 2, or 3, indicating the user's choice.
DO	Begin the outer loop clause. This clause displays option messages, which tell the user what the choices are, and gets a response from the user.
CLLCD	Clear the display.
"Press"	The option message for line 1.
1 DISP	Display the message.
" [A] for Apple"	The option message for line 2.
2 DISP	Display the message.
" [B] for Banana"	The option message for line 3.
3 DISP	Display the message.
" [C] for Cherry"	The option message for line 4.
4 DISP	Display the message.
DO UNTIL KEY END	This inner indefinite loop repeats until the user presses a key. The command KEY returns 0 if no key was pressed, or a string (representing the key) and 1 if a key was pressed. When the loop ends, the string is left on the stack.
UNTIL	Begin the outer test clause. This clause checks whether the key pressed was a defined key.

`{ "A" "B" "C" }`	This list contains the defined keys. There is a one-to-one correspondence between the defined keys and the possible outcomes.
`SWAP POS`	Match the key string to the list of defined keys. POS (*position*) returns 1 if the key string is `"A"`, 2 if the key string is `"B"`, 3 if the key string is `"C"`, or 0 if no match occurs.
`IF DUP`	Make a copy of the position to use as a flag. If the position is 1, 2, or 3, execute the THEN clause. If the position is 0, execute the ELSE clause.
`THEN 1`	The key was defined, so put a true flag on the stack.
`ELSE`	The key was undefined, so display an error message and beep.
`CLLCD "Bad key" 1 DISP`	Display an error message.
`440 .1 BEEP`	Sound a tone.
`1 WAIT`	Wait 1 second.
`END`	End the IF ... THEN ... ELSE ... END structure. If the key was defined, the position and a true flag are on the stack. If the key was undefined, only the position (which is also a false flag) is on the stack.
`END`	End the outer indefinite loop. If the key was defined, the loop ends with the position on the stack. If the key was undefined, the loop clause is repeated.
`GET`	Given the list of possible outcomes and a position, get the correponding outcome.

EVAL   Evaluate the outcome. In this example, EVAL
                        has no effect because the outcome is a string. In
                        a more realistic example, the outcome might be
                        a program (possibly stored in a variable), so
                        EVAL would be needed.

CLMF   Enable the normal stack display.

When this sequence is executed, the user sees the option messages.

```
Press
 [A] for Apple
 [B] for Banana
 [C] for Cherry
```

If the user presses a key other than [A], [B], or [C], a beep sounds and
the error message appears for 1 second.

```
Bad key

```

Then the option messages reappear. When the user presses [A], [B], or
[C], the string "Apple", "Banana", or "Cherry" is returned to
level 1.

By modifying the list of possible outcomes, the option messages, and
the list of defined keys, you can make this sequence more significant
than putting a string on the stack. More generally, by using local vari-
ables and putting this sequence inside a local-variable structure, you
can make the following program.

```
« → keys m1 m2 m3 m4
  « DO CLLCD
      m1 1 DISP
      m2 2 DISP
      m3 3 DISP
      m4 4 DISP
      DO UNTIL KEY END
    UNTIL keys SWAP POS
    IF DUP
    THEN 1
    ELSE CLLCD "Bad key" 1 DISP
      440 .1 BEEP 1 WAIT
    END
    END
  »
  GET EVAL CLMF
»
```

If you store this program in a variable named KEY?, you could perform the example above by executing

```
{ "Apple" "Banana" "Cherry" }
{ "A" "B" "C" }
"Press"
" [A] for Apple"
" [B] for Banana"
" [C] for Cherry"
KEY?
```

28

Programming Examples

This chapter contains 20 programs for your HP-28S. These programs are useful and, more importantly, they demonstrate a variety of programming techniques. For each program you'll find the following information.

- Stack Diagram. A stack diagram is a two-column table showing "Arguments" and "Results". "Arguments" shows what must be on the stack before the program is executed; "Results" shows what the program leaves on the stack.

  The stack diagram doesn't show everything; a program that changes user memory or displays objects might have no effect on the stack.

- Techniques. This is the most interesting part. When you understand how a technique is used in this chapter, you can use it in your own programs.

- Required Programs. Some programs call others as subroutines. You can enter the required programs and the calling program in any order, but you must enter all of them before executing the calling program.

- Program and Comments. This chapter formats the program listing to show a program's structure and process. You don't need to follow the format of the listing when you enter a program. However, be sure to key in spaces where they appear in the listing or between objects appearing on separate lines.

  You can key in a program character by character, or you can use the menus to key it in command by command. It makes no difference as long as the result matches the listing.

When you key in the program you can omit all closing parentheses and delimiters *that appear at the very end of the program*; when you press [ENTER] the closing parentheses and delimiters are added for you.

■ Example. The examples assume STD display format. To select STD display format, press STD [ENTER] or use the MODE menu.

The most important technique demonstrated in this chapter is *structured programming*: small programs used to build other programs. The following programs are used in other programs.

■ BOXS is used in BOXR.

■ MULTI is used in EXCO.

■ PAD and PRESERVE are used in BDISP.

■ ΣGET is used in ΣX2, ΣY2, and ΣXY.

■ SORT and LMED are used in MEDIAN.

Box Functions

This section contains two programs:

■ BOXS calculates the total surface area of a box.

■ BOXR uses BOXS to calculate the ratio of surface to volume for a box.

BOXS (Surface of a Box)

Given the height, width, and length of a box, calculate the total area of its six sides.

Arguments	Results
3 : *height*	3 :
2 : *width*	2 :
1 : *length*	1 : *area*

Techniques:

■ Local-variable structure. Local variables allow you to assign names to arguments without conflicting with global variables. Like global variables, local variables are convenient because you can use arguments any number of times without tracking their positions on the stack; unlike global variables, local variables disappear when the program structure that creates them is done.

A local-variable structure has three parts.

1. A command named "→ ". When you key in this command, remember to put spaces before and after it. (Like any command, → is spelled using normal characters and is recognized only when it's set off by spaces. Don't confuse this one-character command with delimiters like # or «.)

2. One or more names.

3. A procedure (expression, equation, or program) that includes the names. This procedure is called the *defining* procedure.

When a local-variable structure is evaluated, a local variable is created for each name. The values for the local variables are taken from the stack. The defining procedure is then evaluated, substituting the values of the local variables.

To appreciate the power of local variables, compare the version of BOXS given below with the version that appears on page 244.

■ User function. This type of program works in either RPN or algebraic syntax. A user function is a program that consists solely of a local-variable structure and returns exactly one result.

Program	**Comments**
«	Begin the program.
→ h w l	Create local variables for height, width, and length. By convention, lower-case letters are used. The values are taken from the stack (in RPN) or from the arguments to the user function (in algebraic syntax).

Program

```
'2*(h*w+h*1+w*1)'
```

```
»
```

[ENTER]
['] BOXS [STO]

Comments

The defining expression for the surface area. Evaluating the user function causes evaluation of this expression, returning the area to the stack.

End the program.
Put the program on the stack.
Store the program as BOXS.

Example. One of the advantages of user functions is that they work in either RPN or algebraic syntax. Calculate the surface of a box 12 inches high, 16 inches wide, and 24 inches long; make the calculation first in RPN and then in algebraic syntax.

For the RPN version, first enter the height and width.

[USER]
12 [ENTER]
16 [ENTER]

```
3:
2:            12
1:            16
 BOXS
```

Then key in the length and execute BOXS.

24 BOXS

```
3:
2:
1:          1728
 BOXS
```

The surface area is 1728 square inches.

Now try the algebraic version.

['] BOXS [(] 12,16,24 [EVAL]

```
3:
2:          1728
1:          1728
 BOXS
```

Again, the surface area is 1728.

BOXS Without Local Variables

The following program uses only stack operations to calculate the surface of a box. Compare this program with BOXS.

Arguments	Results
3: *height*	3:
2: *width*	2:
1: *length*	1: *area*

Program	Comments
«	Begin the program.
DUP2 *	Calculate *wl*.
ROT	Move *w* to level 1.
4 PICK	Copy *h* to level 1.
*	Calculate *wh*.
+	Calculate *wl* + *wh*.
ROT ROT	Move *h* and *l* to levels 2 and 1.
*	Calculate *hl*.
+	Calculate *wl* + *wh* + *hl*.
2 *	Calculate 2(*wl* + *wh* + *hl*).
»	End the program.

Because this version of BOXS isn't a user function, it can't be used in algebraic syntax.

BOXR (Ratio of Surface to Volume of a Box)

Given the height, width, and length of a box, calculate the ratio of its surface to its volume.

Arguments	Results
3: *height*	3:
2: *width*	2:
1: *length*	1: *area/volume*

Techniques:

■ Nested user functions. BOXR is a user function whose defining expression uses BOXS in its calculation. In turn, BOXR could be used to define other user functions.

Recall that BOXS was defined using *h*, *w*, and *l* as local variables, and note below that BOXS takes *x*, *y*, and *z* as arguments in the definition for BOXR. It makes no difference if the local variables in the two definitions match, or if they don't match, because each set of local variables is independent of the other. However, it's essential that local variables be consistent within a single definition.

Program	Comments
«	Begin the program.
→ x y z	Create local variables for height, width, and length. This program uses *x*, *y*, and *z*, rather than *h*, *w*, and *l*.
'BOXS(x,y,z)	Begin the defining expression with the user function BOXS.
/(x*y*z)'	Divide by the volume of the box.
»	End the program.
ENTER	Put the program on the stack.
' BOXR STO	Store the program as BOXR.

Example. Calculate the ratio of surface to volume for a box 9 inches high, 18 inches wide, and 21 inches long; make the calculation first in RPN and then in algebraic syntax.

For the RPN version, first enter the height and width.

USER
9 ENTER
18 ENTER

```
3:
2:                        9
1:                       18
 BOXR  BOXS
```

Then key in the length and execute BOXR.

21 BOXR

```
3:
2:
1:           .428571428571
 BOXR  BOXS
```

The ratio is .428571428571.

Now try the algebraic version.

' BOXR () 9,18,21 EVAL

```
3:
2:           .428571428571
1:           .428571428571
 BOXR  BOXS
```

Again, the ratio is .428571428571.

Fibonacci Numbers

Given an integer n, calculate the nth Fibonacci number F_n, where

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

This section includes two programs, each demonstrating an approach to this problem.

■ FIB1 is a user function that is defined *recursively*—its defining expression contains its own name. FIB1 is short, easy to understand, and can be used with symbolic arguments.

■ FIB2 is a user function defined with a program. It executes faster than FIB1, but cannot be used with symbolic arguments.

FIB1 (Fibonacci Numbers, Recursive Version)

Arguments	Results
1: n	1: F_n

Techniques:

- IFTE (If-Then-Else function). The defining expression for FIB1 contains the conditional function IFTE, which can be used in either RPN or algebraic syntax. (FIB2 uses the program structure IF ... THEN ... ELSE ... END.)

- Recursion. The defining expression 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
«	Begin the program.
→ n	Define a local variable.
'	Begin the defining expression.
IFTE(n≤1,	If $n \leq 1$,
n,	Then $F_n = n$;
FIB1(n-1)+FIB1(n-2))	Else $F_n = F_{n-1} + F_{n-2}$.
'	End the defining expression.
»	End the program.
[ENTER]	Put the program on the stack.
['] FIB1 [STO]	Store the program as FIB1.

Example. Calculate F_6 using RPN syntax and F_{10} using algebraic syntax.

First calculate F_6 using RPN.

[USER]
6 FIB1

```
3:
2:
1:                          8
FIB1 BOXR BOXS
```

Next calculate F_{10} using algebraic syntax.

['] FIB1 [(] 10 [EVAL]

```
3:
2:                          8
1:                         55
FIB1 BOXR BOXS
```

FIB2 (Fibonacci Numbers, Loop Version)

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's.

Program	Comments
«	Begin the program.
→ n	Create a local variable.
«	Begin the defining program.
IF n 1 ≤	If n ≤ 1,
THEN n	Then $F_n = n$;
ELSE	Begin ELSE clause.
0 1	Put F_0 and F_1 on the stack.
2 n	From 2 to n,
START	Do the following loop:
DUP	Make a copy of the latest F (initially F_1).
ROT	Move the previous F (initially F_0) to level 1.
+	Calculate the next F (initially F_2).
NEXT	Repeat the loop.
SWAP DROP	Drop F_{n-1}.
END	End ELSE clause.
»	End the defining program.
»	End the program.
ENTER	Put the program on the stack.
' FIB2 STO	Store the program as FIB2.

Example. Calculate F_6 and F_{10}. Note that FIB2 is faster than FIB1.
Calculate F_6.

USER

6 FIB2

```
3:
2:
1:            8
FIB2 FIB1 BOXR BOXS
```

Calculate F_{10}.

10 FIB2

```
3:
2:            8
1:           55
FIB2 FIB1 BOXR BOXS
```

Comparison of FIB1 and FIB2

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.
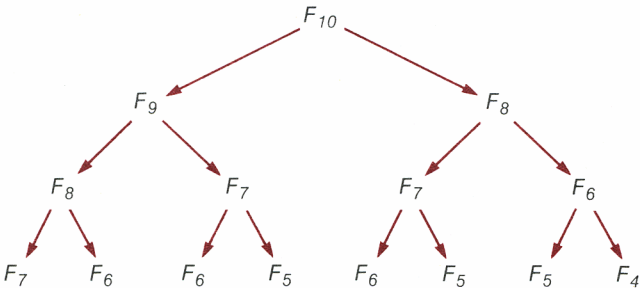
Single-Step Execution

It's easier to understand how a program works if you execute it step by step, seeing the effect on the stack of each step. Doing this can help you "debug" your own programs or help you understand programs written by others.

This section shows you how to execute FIB2 step by step, but you can apply these rules to any program. The general rules are:

1. Use VISIT to insert the command HALT in the program. Place HALT where you want to begin single-step execution. (You'll see how the position of HALT within FIB2 affects execution.)

2. Execute the program. When the HALT command is executed, the program stops (indicated by the "stopsign" annunciator).

3. Select the PROGRAM CONTROL menu.

4. Press SST once to see the next program step displayed and then executed.

   You can now:

   ■ Keep pressing SST to display and execute sequential steps.

   ■ Press ■CONT to continue normal execution.

   ■ Press KILL to abandon further program execution.

5. When you want the program to run normally again, use VISIT to remove HALT from the program.

For the first example, insert HALT as the first command in FIB2. Clear the stack and select the USER menu.

■ CLEAR
USER

```
3:
2:
1:
FIB2 FIB1 BOXR BOXS
```

Use VISIT to return FIB2 to the command line.

' FIB2 ■ VISIT

```
■  → n
  «
    IF n 1 ≤
    THEN n
```

Insert the HALT command.

▶ INS ■ CONTRL HALT

```
« HALT ◀▶ n
  «
    IF n 1 ≤
SST HALT ABORT KILL WAIT KEY
```

Store the edited version of FIB2.

ENTER

```
3:
2:
1:
SST HALT ABORT KILL WAIT KEY
```

Calculate F_1. At first, nothing happens except that the O annunciator appears.

USER
1 FIB2

```
3:
2:
1:                          1
FIB2 FIB1 BOXR BOXS
```

Select the PROGRAM CONTROL menu and execute SST (*single-step*). Watch the top line of the display to see the first step displayed before it's executed.

■ CONTRL
SST

```
→ n
2:
1:                          1
SST HALT ABORT KILL WAIT KEY
```

Note that → n constitutes one step; "step" is a logical unit rather than simply the next object in the program.

Look at the general rules at the beginning of this section. Now you can choose one of the three alternatives described in step 4.

For this example, press ░SST░ repeatedly until the ⭕ annunciator disappears, indicating that FIB2 is completed. (These single-steps not shown here.)

The calculation for F_1 executes only the THEN clause in FIB2. For the second example, execute 3 FIB2 and single-step through the calculation for F_3. This executes the ELSE clause, including the START ... NEXT loop. You'll see that, for $n = 3$, the START ... NEXT loop is executed twice.

For the third example, suppose you want to single-step the START ... NEXT loop as a whole—seeing the stack before each iteration of the loop, but not single-stepping all the steps in FIB2 or in the loop itself. To do so, move the HALT command inside the loop. Then FIB2 won't halt until it reaches the loop, and you can use ⬛⎯CONT⎯ (*continue*) to execute the loop one iteration at a time.

Use VISIT to return FIB2 to the command line.

⎯USER⎯
▢ FIB2 ⬛⎯VISIT⎯

```
⬛ HALT → n
   «
     IF n 1 ≤
     THEN n
```

Use the cursor menu keys to delete HALT. Then insert HALT as shown (following the START command).

```
IF n 1 ≤
THEN n
ELSE 0 1 2 n
     START HALT♦DUP R...
```

Store the edited version of FIB2.

⎯ENTER⎯

```
3:
2:
1:
FIB2 FIB1 BOXR BOXS
```

Start the calculation for F_3. FIB2 will halt before performing the loop.

3 FIB2

```
3:
2:            0
1:            1
FIB2 FIB1 BOXR BOXS
```

Continue execution of the loop. FIB2 will halt before performing the loop a second time.

■ CONT

```
3:
2:                          1
1:                          1
FIB2 FIB1 BOXR BOXS
```

Continue execution of the loop. Because this is the last iteration of the loop, FIB2 will execute to completion.

■ CONT

```
3:
2:
1:                          2
FIB2 FIB1 BOXR BOXS
```

When you're done experimenting with FIB2, don't forget to use VISIT to remove the HALT command.

Expanding and Collecting Completely

This section contains two programs:

■ MULTI repeats a program until the program has no effect.
■ EXCO uses MULTI to expand and collect completely.

MULTI (Multiple Execution)

Given an object and a program that acts on the object, apply the program to the object repeatedly until the object is unchanged.

Arguments	Results
2: *object*	2:
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 handy to store an object in a local variable when you don't know beforehand how many copies you'll need.

MULTI demonstrates one of the differences between global and local variables: if a global variable contains a name or program, the contents of the variable are evaluated when the name is evaluated; but the contents of a local variable are always simply recalled. Consequently, MULTI uses the local name to put the program argument on the stack and then executes an explicit EVAL command to evaluate the program.

Program	Comments
«	Begin the program.
→ P	Create a local variable *p* that contains the program argument.
«	Begin the defining program.
DO	Begin the DO clause.
DUP	Make a copy of the object.
P EVAL	Apply the program to the object, returning a new version. (The EVAL command is necessary to execute the program because local variables always return their contents to the stack unevaluated.)
UNTIL	Begin the UNTIL clause.
DUP	Make a copy of the new version of the object.
ROT	Move the old version to level 1.
SAME	Test whether the old version and the new version are the same.

Program	Comments
END	End the UNTIL clause.
»	End the defining program.
»	End the program.
[ENTER]	Put the program on the stack.
['] MULTI [STO]	Store the program as MULTI.

Example. MULTI is demonstrated in the next program.

EXCO (Expand and Collect Completely)

Given an algebraic object, execute EXPAN repeatedly until the algebraic doesn't change, then execute COLCT repeatedly until the algebraic doesn't change. In some cases the result will be a number.

Arguments	Results
1 : ' *algebraic* '	1 : ' *algebraic* '
1 : ' *algebraic* '	1 : *z*

Techniques:

■ Structured programming. EXCO calls the program MULTI twice. Even if you don't use MULTI anywhere else, the efficiency of repeating all the commands in MULTI by simply including its name a second time justifies writing MULTI as a separate program.

Required Programs:

■ MULTI (page 253) repeatedly executes the programs that EXCO provides as arguments.

Program	Comments
«	Begin the program.
« EXPAN »	Put EXPAN on the stack.
MULTI	Execute EXPAN until the algebraic object doesn't change.
« COLCT »	Put COLCT on the stack.
MULTI	Execute COLCT until the algebraic object doesn't change.
»	End the program.
[ENTER]	Put the program on the stack.
['] EXCO [STO]	Store the program as EXCO.

Example. Expand and collect completely the expression

$$3x\,(4y + z) + (8x - 5z)^2.$$

Enter the expression.

[USER]
['] 3 [×] X [×]
[(] 4 [×] Y [+] Z ▮[)] [+]
[(] 8 [×] X [−] 5 [×] Z ▮[)] ▮[^] 2
[ENTER]

```
2:
1:  '3*X*(4*Y+Z)+(8*X-5*
    Z)^2'
EXCO MULT FIB2 FIB1 BOXR BOXS
```

Expand and collect completely.

EXCO

```
2:
1:  '12*X*Y-77*X*Z+64*X^
    2+25*Z^2'
EXCO MULT FIB2 FIB1 BOXR BOXS
```

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.

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)

Convert an object to a string and, if the string contains fewer than 23 characters, add spaces to the beginning.

When a short string is displayed by using 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 is 23 characters long, it appears *right-justified*: its last character appears at the right end of the display.

PAD has no effect on strings that are longer than 22 characters.

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 concatenate two strings.

Program	Comments
«	Begin the program.
→STR	Make sure the object is in string form. (Strings are unaffected by this command.)
WHILE	Begin WHILE clause.
DUP SIZE 23 <	Does the string contain fewer than 23 characters?
REPEAT	Begin REPEAT clause.
" " SWAP +	Add a leading space.
END	End REPEAT clause.
»	End the program.
ENTER	Put the program on the stack.
' PAD STO	Store the program as PAD.

Example. PAD is demonstrated in the program BDISP.

PRESERVE (Save and Restore Previous Status)

Given a program on the stack, store the current status, execute the program, and then restore the previous status.

Arguments	Results
1 : « *program* »	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 just to remove the object from the stack briefly; its defining program does little except evaluate the program argument on the stack.

Program	Comments
«	Begin the program.
RCLF	Recall a 64-bit binary integer representing the status of all 64 user flags.
→ f	Store the binary integer in a local variable f.
«	Begin the defining program.
EVAL	Execute the program argument.
f STOF	Restore the status of all 64 user flags.
»	End the defining program.
»	End the program.
ENTER	Put the program on the stack.
['] PRESERVE [STO]	Store the program as PRESERVE.

Example. PRESERVE is demonstrated in the program BDISP.

BDISP (Binary Display)

Display a number in HEX, DEC, OCT, and BIN bases.

Arguments	Results
1: # n	1: # n
1: n	1: n

Techniques:

- IFERR ... THEN ... END (error trap). To accomodate real numbers, 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. Because no action is required when an error occurs, the THEN clause contains no commands.

- Enabling LAST. In case an error occurs, LAST must be enabled to return the argument to the stack. BDISP sets flag 31 to programmatically enable the LAST recovery feature.

- FOR ... NEXT loop (definite loop with counter). BDISP executes a loop from 1 to 4, each time displaying n in a different base on a different line.

  The loop counter (named j in this program) is a local variable. It's created by the FOR ... NEXT program structure (rather than by a → command) and it's automatically incremented by NEXT.

- Subprograms. BDISP demonstrates three uses for subprograms.

  1. BDISP contains a main subprogram and a call to PRESERVE. The main subprogram goes on the stack and is evaluated by PRESERVE.

  2. When BDISP creates a local variable for n, the defining program is a subprogram.

  3. There are four subprograms that "customize" the action of the loop. Each subprogram contains a command to change the binary base, and each iteration of the loop executes one of these subprograms.

Required Programs:

- PAD (page 257) expands a string to 23 characters so that DISP shows it right-justified.

- PRESERVE (page 258) stores the current status, executes the main subprograms and restores the status.

Program	Comments
«	Begin the program.
«	Begin the main subprogram.
DUP	Make a copy of n.
31 SF	Set flag 31 to enable LAST.
IFERR	Begin error trap.
R→B	Convert n to a binary integer.
THEN	If an error occured,
END	Do nothing (no commands in THEN clause).
→ n	Create a local variable n.
«	Begin the defining program.
CLLCD	Clear the display.
« BIN »	Subprogram for BIN.
« OCT »	Subprogram for OCT.
« DEC »	Subprogram for DEC.
« HEX »	Subprogram for HEX.
1 4	First and last counter values.
FOR j	Start loop with counter j.
EVAL	Evaluate one of the base subprograms (initially the one for HEX).
n →STR	Make a string showing n in the current base.
PAD	Pad the string to 23 characters.
j DISP	Display the string in the jth line.
NEXT	Increment j and repeat the loop.
»	End the defining program.
»	End the main subprogram.
PRESERVE	Store the current status, execute the main subprogram, and restore the status.
»	End the program.
ENTER	Put the program on the stack.
' BDISP STO	Store the program as BDISP.

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 BINARY menu.

■ CLEAR
■ BINARY

```
3:
2:
1:
DEC■ HEX  OCT  BIN STWS RCWS
```

Make sure the current base is DEC and key in # 100.

DEC
100 ENTER

```
3:
2:
1:              # 100d
DEC■ HEX  OCT  BIN STWS RCWS
```

Execute BDISP. (Don't switch menus, since you'll want to see the BI-NARY menu in the next step.)

BDISP ENTER

```
                    # 64h
                   # 100d
                   # 144o
            # 1100100b
```

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

ON

```
3:
2:
1:              # 100d
DEC■ HEX  OCT  BIN STWS RCWS
```

Although the main subprogram left the calculator in BIN base, PRE-SERVE restored DEC base.

To check that BDISP also works for real numbers, try 144.

USER
144 BDISP

```
                    # 90h
                   # 144d
                   # 220o
          # 10010000b
```

Summary Statistics

For paired-sample statistics it's often useful to calculate the sum of the squares (Σx^2 and Σy^2) and the sum of the products (Σxy) of the two variables. This section contains five programs:

- SUMS creates a variable ΣCOV that contains the covariance matrix for the current statistics matrix ΣDAT.

- ΣGET extracts a number from the specified position in ΣCOV.

- ΣX2 uses ΣGET to extract Σx^2 from ΣCOV.

- ΣY2 uses ΣGET to extract Σy^2 from ΣCOV.

- ΣXY uses ΣGET to extract Σxy from ΣCOV.

If ΣDAT contains n columns, ΣCOV is an $n \times n$ matrix. The programs ΣX2, ΣY2, and ΣXY refer to ΣPAR (*statistics parameters*) to determine which columns contain the x data (called C_1) and the y data (called C_2).

Techniques:

- Matrix operations. These programs demonstrate how to transpose a matrix, how to multiply two matrices, and how to extract one element from a matrix.

- Programs usable in algebraic objects. Because ΣX2, ΣY2, and ΣXY conform to algebraic syntax (no arguments from the stack, one result put on the stack), you can use their names like ordinary variables in an expression or equation.

- ΣPAR convention. Several paired-sample statistics commands use a variable named ΣPAR to specify a pair of columns in ΣDAT. ΣPAR contains a list with four numbers, the first two specifying columns. (The other two numbers are the slope and intercept from linear regression.)

  SUMS ensures that ΣPAR exists by executing 0 PREDV DROP; the command PREDV (*predicted value*) creates ΣPAR with default values if ΣPAR doesn't already exist, and DROP removes the predicted value computed for 0.

  ΣX2, ΣY2, and ΣXY use the values stored in ΣPAR to determine which element to extract from ΣCOV.

SUMS (Summary Statistics Matrix)

Create a variable ΣCOV that contains the covariance matrix of the statistics matrix ΣDAT.

As an example, if ΣDAT is the $n \times 2$ matrix

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

then ΣCOV will contain the covariance matrix

$$\begin{bmatrix} \Sigma x^2 & \Sigma xy \\ \Sigma xy & \Sigma y^2 \end{bmatrix}$$

Arguments	Results
1 :	1 :

Program	Comments
«	Begin the program.
RCLΣ	Recall the contents of the $n \times m$ statistics matrix ΣDAT.
DUP	Make a copy.
TRN	Transpose the matrix. The result is an $m \times n$ matrix.
SWAP *	Multiply the matrices to produce the $m \times m$ covariance matrix. (Without swapping the matrices, the product would be an $n \times n$ matrix.)
'ΣCOV' STO	Store the covariance matrix in a variable ΣCOV.
0 PREDV DROP	Make sure ΣPAR exists.
»	End the program.
[ENTER]	Put the program on the stack.
['] SUMS [STO]	Store the program as SUMS.

ΣGET (Get an Element of ΣCOV)

Given p and q, each indicating either the first or second *position* in ΣPAR, extract the *rs* element from ΣCOV, where r and s are the corresponding first or second *elements* in ΣPAR.

ΣGET is called by ΣX2, ΣY2, and ΣXY with the following arguments.

- For ΣX2, $p = 1$ and $q = 1$.
- For ΣY2, $p = 2$ and $q = 2$.
- For ΣXY, $p = 1$ and $q = 2$.

Arguments	Results
2 : *1 or 2*	2 :
1 : *1 or 2*	1 : *rs element of ΣCOV*

Program

```
«
  ΣCOV

  ΣPAR

  DUP
  5 ROLL
  GET
  SWAP
  4 ROLL
  GET
  2 →LIST
  GET
»
```

[ENTER]
['] ΣGET [STO]

Comments

Begin the program.
Put the covariance matrix on the stack.
Put the list of statistics parameters on the stack.
Make a copy.
Move p to level 1.
Get r, the pth element in ΣPAR.
Move ΣPAR to level 1.
Move q to level 1.
Get s, the qth element in ΣPAR.
Put { r, s } on the stack.
Get the *rs* element from ΣCOV.
End the program.

Put the program on the stack.
Store the program as ΣGET.

ΣX2 (Sum of Squares of x)

Calculate Σx^2, where the x's are the elements of C_1 (the column specified by the first parameter in ΣPAR).

Arguments	Results
1 :	1 : Σx^2

Program	Comments
«	Begin the program.
1 1	Specify C_1 twice.
ΣGET	Extract Σx^2.
»	End the program.
ENTER	Put the program on the stack.
' ΣX2 STO	Store the program as ΣX2.

ΣY2 (Sum of Squares of y)

Calculate Σy^2, where the y's are the elements of C_2 (the column specified by the second parameter in ΣPAR).

Arguments	Results
1 :	1 : Σy^2

Program	Comments
«	Begin the program.
2 2	Specify C_2 twice.
ΣGET	Extract Σy^2.
»	End the program.
ENTER	Put the program on the stack.
' ΣY2 STO	Store the program as ΣY2.

ΣXY (Sum of Products of x and y)

Calculate Σxy, where the x's and y's are corresponding elements of C_1 and C_2 (the columns specified by the first and second parameters in ΣPAR).

Arguments	Results
1 :	1 : Σxy

Program	Comments
«	Begin the program.
1 2	Specify C_1 and C_2.
ΣGET	Extract Σxy.
»	End the program.
[ENTER]	Put the program on the stack.
['] ΣXY [STO]	Store the program as ΣXY.

Example. Calculate ΣX2, ΣY2, and ΣXY for the following statistics data:

$$\begin{bmatrix} 18 & 12 \\ 4 & 7 \\ 3 & 2 \\ 11 & 1 \\ 31 & 48 \\ 20 & 17 \end{bmatrix}$$

The general steps are as follows.

1. Enter the statistical data.

2. Execute SUMS to create the covariance matrix ΣCOV.

3. Execute ΣX2, ΣY2, and ΣXY.

4. If ΣDAT contains more than two columns (that is, if each data point contains more than two variables):

   a. Execute COLΣ to specify new values for C_1 and C_2. The values are stored in ΣPAR.

   b. Execute ΣX2, ΣY2, and ΣXY.

Now try the example given above.

Clear the stack, select the STAT menu, and clear ΣDAT.

■ [CLEAR]
■ [STAT]
  CLΣ

```
3:
2:
1:
Σ+  Σ-  NΣ  CLΣ  STOΣ  RCLΣ
```

Enter the data and then check that you entered all six data points.

[I] 18,12 Σ+
[I] 4,7   Σ+
[I] 3,2   Σ+
[I] 11,1   Σ+
[I] 31,48 Σ+
[I] 20,17 Σ+
   NΣ

```
3:
2:
1:                          6
Σ+  Σ-  NΣ  CLΣ  STOΣ  RCLΣ
```

Drop the number of data points.

[DROP]

```
3:
2:
1:
Σ+  Σ-  NΣ  CLΣ  STOΣ  RCLΣ
```

Create the covariance matrix ΣCOV.

USER
SUMS

```
3:
2:
1:
ΣPAR ΣCOV ΣDAT ΣXY ΣY2 ΣX2
```

Calculate Σx^2.

ΣX2

```
3:
2:
1:                     1831
ΣPAR ΣCOV ΣDAT ΣXY ΣY2 ΣX2
```

Calculate Σy^2.

ΣY2

```
3:
2:                     1831
1:                     2791
ΣPAR ΣCOV ΣDAT ΣXY ΣY2 ΣX2
```

Calculate Σxy.

ΣXY

```
3:                     1831
2:                     2791
1:                     2089
ΣPAR ΣCOV ΣDAT ΣXY ΣY2 ΣX2
```

If the statistics matrix had more than two columns, you could specify new values for C_1 and C_2. For practice, specify $C_1 = 1$ and $C_2 = 2$ (the current values).

The command COLΣ is available in the STAT menu, but here it's easier to spell out the command name and stay in the USER menu.

1 ENTER
2 COLΣ ENTER

```
3:                     1831
2:                     2791
1:                     2089
ΣPAR ΣCOV ΣDAT ΣXY ΣY2 ΣX2
```

You could now execute ΣX2, ΣY2, and ΣXY for the new pair of columns C_1 and C_2.

Don't forget to execute SUMS again whenever you add or delete data from the statistics matrix ΣDAT.

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 a list into ascending order.

Arguments	Results
1: { *list* }	1: { *sorted list* }

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 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.

- Nested local-variable structures. SORT contains two local-variable structures, the second inside the defining program of the first. This nesting is done for convenience; it's easier to create the first local variable as soon as its value is computed, thereby removing its value from the stack, rather than computing both values and creating both local variables at once.

- 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
«	Begin the program.
DUP SIZE 1 - 1	From the next-to-last position to the first position,
FOR j	Begin the outer loop with counter j.
1 j	From the first position to the jth position,
FOR k	Begin the inner loop with counter k.
k GETI → n1	Get the kth number in the list and store it in a local variable n_1.
«	Begin outer defining program.
GETI → n2	Get the next number in the list and store it in a local variable n_2.
«	Begin inner defining program.
DROP	Drop the counter.
IF n1 n2 >	If the two numbers are in the wrong order,
THEN	Then do the following:
k n2 PUTI	Put the second one back in the kth position.
n1 PUT	Put the kth one back in the next position.
END	End of THEN clause.
»	End inner defining program.
»	End outer defining program.
NEXT	Increment k and repeat the inner loop.
-1 STEP	Decrement j and repeat the outer loop.
»	End the program.
[ENTER]	Put the program on the stack.
['] SORT [STO]	Store the program as SORT.

Example.

Sort the list { 8, 3, 1, 2, 5 }.

USER

[ ] 8,3,1,2,5 SORT

```
3:
2:
1:          { 1 2 3 5 8 }
SORT ΣPAR ΣCOV ΣDAT ΣXY ΣY2
```

LMED (Median of a List)

Given a sorted list, calculate 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 non-integer, FLOOR and CEIL return successive integers that bracket the non-integer.

Program	Comments
«	Begin the program.
DUP SIZE	The size of the list.
1 + 2 /	The center position in the list (fractional for even-sized lists).
→ p	Store the center position in local variable p.
«	Begin the defining program.
DUP	Make a copy of the list.
p FLOOR GET	Get the number at or below the center position.
SWAP	Move the list to level 1.
p CEIL GET	Get the number at or above the center position.

Program	Comments
+ 2 /	The average of the two numbers at or near the center position.
»	End the defining program.
»	End the program.
ENTER	Put the program on the stack.
['] LMED STO	Store the program as LMED.

Example.

Calculate the median of the list you sorted using SORT.

USER
LMED

```
3:
2:
1:                                    3
LMED SORT ΣPAR ΣCOV ΣDAT ΣXY
```

LMED is called by MEDIAN.

MEDIAN (Median of Statistics Data)

Return 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 mth 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 270) arranges a list in ascending order.
- LMED (page 272) calculates the median of a sorted list.

Program	Comments
«	Begin the program.
RCLΣ	Put a copy of the current statistics matrix ΣDAT on the stack for safekeeping.
DUP SIZE	Put the list { n m } on the stack, where n is the number of rows in ΣDAT and m is the number of columns.
LIST→ DROP	Put n and m on the stack. Drop the list size.
→ n m	Create local variables for n and m.
«	Begin the defining program.
'ΣDAT' TRN	Transpose ΣDAT. Now n is the number of columns in ΣDAT and m is the number of rows.
1 m	The first and last rows.
FOR j	For each row, do the following:
Σ-	Extract the last row in ΣDAT. Initially this is the mth row, which corresponds to the mth column in the original ΣDAT.
ARRY→ DROP	Put the row elements on the stack. Drop the index list { n }, since n is already stored in a local variable.
n →LIST	Make an n-element list.
SORT	Sort the list.
LMED	Calculate the median of the list.

Program	Comments
`j ROLLD`	Move the median to the proper stack level.
`NEXT`	Increment *j* and repeat the loop.
`m 1 →LIST`	Make the list { *m* }.
`→ARRY`	Combine all the medians into an *m*-element vector.
`»`	End the defining program.
`SWAP`	Move the orginal ΣDAT to level 1.
`STOΣ`	Restore ΣDAT to its previous value.
`»`	End the program.
ENTER	Put the program on the stack.
['] MEDIAN [STO]	Store the program as MEDIAN.

Example. Calculate the median of the data on page 268. (This example assumes you've keyed in the data.) There are two columns of data, so MEDIAN will return a two-element vector.

Calculate the median.

USER
MEDI

```
3:
2:
1:            [ 14.5 9.5 ]
ΣDAT MEDI LMED SORT ΣPAR ΣCOV
```

The medians are 14.5 for the first column and 9.5 for the second column.

Changing Directories

This section contains two programs:

■ UP gives you a menu of parent directories.
■ DOWN gives you a menu of subdirectories.

These programs have no utility for those who always remember their entire directory structure and know exactly where they are at all times. For those who occasionally become confused, these programs are helpful.

UP (Move to a Parent Directory)

Create a menu that contains the names of the parent directory, its parent directory, and so on, back to the HOME directory.

Arguments	Results
1 :	1 :

Techniques:

■ List of parent directories. UP uses PATH to return the names of the current directory and all parent directories.

■ Subset of a list. UP uses SUB to remove the name of the current directory from the PATH list.

■ Custom menu. UP uses MENU to create a custom menu of parent directories from the modified PATH list.

Program	Comments
«	Begin the program.
PATH	Put the path list on the stack.
1	Put 1 on the stack.
OVER SIZE 1 −	Put *size* − 1 on the stack.
SUB	Create a subset of the PATH list that includes all names but the last name (the current directory).
MENU	Create a menu of parent directories.
»	End the program.
[ENTER]	Put the program on the stack.
['] UP [STO]	Store the program as UP.

Example. From the HOME directory, create a hierarchy of subdirectories D1, D2, and D3; then use UP to move from D3 to D1.

Clear the stack and move to the HOME directory.

■ CLEAR
■ MEMORY   HOME

```
3:
2:
1:
MEM MENU ORDER PATH HOME CRDIR
```

Create a subdirectory D1 and move to it.

`'` D1 CRDIR
D1 ENTER

```
3:
2:
1:
MEM MENU ORDER PATH HOME CRDIR
```

Repeat the process for subdirectories D2 and D3.

`'` D2 CRDIR
D2 ENTER
`'` D3 CRDIR
D3 ENTER

```
3:
2:
1:
MEM MENU ORDER PATH HOME CRDIR
```

Display the menu of parent directories.

UP ENTER

```
3:
2:
1:
HOME  D1   D2
```

Move to the D1 directory.

   D1

```
3:
2:
1:
HOME  D1   D2
```

DOWN (Move to a Subdirectory)

Create a menu that contains the names of all subdirectories of the current directory.

Arguments	Results
1 :	1 :

Techniques:

- List of variables. DOWN uses VARS to return the list of variables and subdirectories in the current directory.

- Error trap. To check whether a name in the VARS list is a directory, DOWN uses the name as an argument to RCL; since directories can't be recalled to the stack, an error occurs if the name is a directory name, and the name is added to the list of directory names.

Program	Comments
«	Begin the program.
VARS	Put on the stack a list of the names of all variables and subdirectories.
→ v	Store the VARS list in a local variable *v*.
«	Begin the defining program.
{ }	Put the list of directory names on the stack (initally empty).
1 v SIZE	Put 1 and *size of v* on the stack.
FOR j	For each name in *v*, do the following:
v j GET	Get the name.
IFERR RCL DROP	Attempt to recall the contents of a variable with that name; if successful, drop the contents.
THEN +	If RCL caused an error, the name must be a directory name, so add the name to the list of directory names.
END	End of the THEN clause and the program structure.
NEXT	Repeat for next name in *v*.
MENU	Create a custom menu for the directory names.
»	End the defining program.
»	End the program.
ENTER	Put the program on the stack.
⌐ DOWN STO	Store the program as DOWN.

Example. In the previous example (page 277) you created a hierarchy of subdirectories D1, D2, and D3, and completed the example with D1 the current directory. For this example, move to D2 and then D3.

Display the menu of subdirectories.
DOWN [ENTER]

```
3:
2:
1:
┌──┬──┬──┬──┬──┐
│D2│  │  │  │  │
└──┴──┴──┴──┴──┘
```

Move down to D2.
D2

```
3:
2:
1:
┌──┬──┬──┬──┬──┐
│D2│  │  │  │  │
└──┴──┴──┴──┴──┘
```

Display the menu of subdirectories.
DOWN [ENTER]

```
3:
2:
1:
┌──┬──┬──┬──┬──┐
│D3│  │  │  │  │
└──┴──┴──┴──┴──┘
```

Move down to D3.
D3

```
3:
2:
1:
┌──┬──┬──┬──┬──┐
│D3│  │  │  │  │
└──┴──┴──┴──┴──┘
```

Appendixes & Indexes

A

Assistance, Batteries, and Service

This appendix contains information to help you when you have problems with your calculator. If you have problems understanding how to use the calculator, and you can't find an appropriate topic in the Table of Contents (page 5) or the Subject Index (page 332), see "Answers to Common Questions" below. If you don't find an answer to your question, you can contact our Calculator Support department, using the address or phone number listed on the inside back cover.

If you need to replace the batteries, see page 286. If your calculator doesn't seem to work properly, see "Determining If the Calculator Requires Service" on page 289. If the calculator does require service, see "Limited One-Year Warranty" on page 291 and "If the Calculator Requires Service" on page 293.

Answers to Common Questions

Q: *The calculator doesn't turn on when I press* ON . *What is wrong?*
A: There may be a simple problem that you can solve immediately, or the calculator may require service. See "Determining If the Calculator Requires Service" on page 289.

Q: *How can I verify that the calculator is operating properly?*
A: Perform the repeating test, as described on page 290.

Q: *How do I clear everything from the calculator's memory?*
A: Press and hold ON INS ▶ , then release, as described in "Clearing All Memory (Memory Reset)" on page 20.

Q: *What do three dots (. . .) mean at the right 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.

Q: *How do I display all of an object?*
A: Use ■ EDIT or ■ VISIT to return the object to the command line, as described in "Editing Exisiting Objects" on page 173. You can then use the cursor keys to display any part of the object. To cancel the edit, press ON .

Q: *What does "object" mean?*
A: "Object" is a general term for almost everything you work with. Numbers, expressions, arrays, programs, and so on, are all types of objects. See "Major Features and Concepts" on page 25 for a brief description of object types, or see chapter 16, "Objects," for a detailed discussion of object types.

Q: *The calculator beeps and displays* Bad Argument Type. *What is wrong?*
A: The objects on the stack aren't the correct type for the command you're attempting. For example, executing STO without a name in level 1 causes this error. Use CATALOG to check the correct arguments for the command, as described in "The Catalog of Commands" on page 31.

Q: *The calculator beeps and displays* Too Few Arguments. *What is wrong?*
A: There are fewer objects on the stack than required by the command you're attempting. For example, executing + with only one number on the stack causes this error. Use CATALOG to check the correct arguments for the command, as described in "The Catalog of Commands" on page 31.

Q: *The calculator beeps and displays an error message different from the two listed above. How do I find out what's wrong?*
A: See appendix A, "Messages," in the Reference Manual.

Q: *How do I turn off the beeper?*
A: Type 51 SF ENTER . This sets flag 51, which disables the beeper.

Q: *How can I print a copy of the display?*
A: Press and hold ON , press L , and release ON .

Q: *The keys from* ⒜ *to* ⒭ *don't work. What is wrong?*
A: You accidentally selected Menu Lock, so the keys from ⒜ to ⒭ select menus unless you press ■ first. To turn off Menu Lock, press ■ [MENUS].

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.

Q: *How can I determine how much memory is left?*
A: Execute MEM [ENTER] to return the number of bytes available in memory.

Q: *Why did the cursor change its appearance?*
A: The cursor indicates the current entry mode. The entry modes are Immediate (empty cursor), Algebraic (partly filled cursor), or Alpha (filled cursor). The shape of the cursor indicates Replace mode (box cursor) or Insert mode (arrow cursor). See "How the Cursor Indicates Modes" on page 172.

Q: *I keyed in a name (or pressed a USER menu key), but the name didn't go on the stack. Why not?*
A: You entered an *unquoted* name, which refers to the *contents* of a variable. To put a name on the stack, press ⒠ first. (See "Quoted and Unquoted Names" on page 57.)

Q: *When I calculate the cube root of* −27, *why isn't the result* −3?
A: Every number has three cube roots, two of which are complex numbers. The HP-28S returns one of the three roots, called the principal value. For positive real arguments the principal value is the real root; for negative real arguments the principal value is one of the complex roots. To calculate the real *b*th root of a real number *a*, key in the following program.

```
« → a b 'SIGN(a)*ABS(a)^INV(b)' »
```

Press ⒠ RROOT [STO] to store the program in a variable RROOT (*real root*). You can then find the real cube root of −27 by typing 27 [CHS] [ENTER] 3 [ENTER] RROOT [ENTER].

Q: *The calculator is slower than usual, and the ⬯ annunciator is blinking. What is happening?*
A: The calculator is in Trace printing mode. Press ⬛ PRINT TRAC to turn off Trace printing mode.

Q: *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 make sure the printer can keep up.

Q: *How can I speed up printing?*
A: If your printer is plugged into an adaptor, the calculator can safely send data at a faster rate. To select faster printing, type 52 SF ENTER. This sets flag 52, which controls the printing speed. When the printer isn't plugged into an adaptor, type 52 CF ENTER to clear flag 52 and return to normal printing speed.

Q: *The printer drops characters or prints ⬛ characters. What is wrong?*
A: The distance or angle between the printer and the calculator may be too large, or there may be an obstruction blocking the transmission. See the printer manual for details about positioning the printer and calculator.

Q: *What is the difference between STO and STORE?*
A: The STO command assigns a specified value to a variable. The STORE menu contains commands that perform *storage arithmetic*, using the value of a variable as an argument and assigning the resulting value to the variable.

Q: *I expected a symbolic result, but I got a numerical result. Why?*
A: There are values assigned to one or more variables. Purge the contents of the variables (see "Purging a Variable" on page 52) and then try again.

Q: *When I press* DRAW *, the display clears, the ((•)) annunciator blinks and then stops, but I don't see any points plotted on the display. Why not?*
A: The calculated values are outside the current plot range. See "Changing the Scale of the Plot" on page 91.

Q: *I evaluated a variable or an expression, and now the calculator doesn't respond. Pressing* [ON] *has no effect. What happened?*

A: You defined a variable in terms of itself, creating a circular definition, and now the calculator is executing an "endless loop." To terminate the loop, perform a System Halt as follows:

1. Press and hold [ON].

2. Press [▲].

3. Release [ON].

Then redefine the variable to remove the circular definition.

If you don't find an answer to your question, you can contact our Calculator Technical Support department, using the address or phone number listed on the inside back cover.

Batteries

The HP-28S is powered by three alkaline batteries. A fresh set of batteries typically will provide approximately six months to one year of use. However, expected battery life depends on how the calculator is used.

Use only fresh N-cell alkaline batteries. Do not use rechargeable batteries.

Low Power Indicator

When the low battery annunciator (▭) comes on, the HP-28S can continue operating for at least 10 hours. If the calculator is turned off when the annunciator first comes on, Continuous Memory will be preserved for approximately one month.

Installing Batteries

If you have just purchased the HP-28S and are installing the batteries for the first time, you can take as long as you'd like to complete these procedures.

However, if you are replacing batteries, you should keep in mind that there is a time limit for completing these procedures if you want to preserve the information you have stored inside the calculator (Continuous Memory). Once the battery compartment is open, you must replace the batteries and close the compartment within one minute to prevent loss of Continuous Memory. Therefore, you should have the new batteries readily at hand before opening the battery compartment. Also, you must make sure the calculator is off during the entire process of changing batteries.

To install batteries:

1. Have three fresh N-cell batteries readily at hand.

2. Open the calculator to expose the keyboard and display. If you are replacing batteries, make sure the calculator is off. *Do not press* ON *until the entire procedure for changing batteries is completed. Changing batteries with the calculator on could erase the contents of Continuous Memory.*

3. Hold the calculator with the battery compartment door facing up. To remove the battery compartment door, slide it towards the back of the calculator (away from the product label).
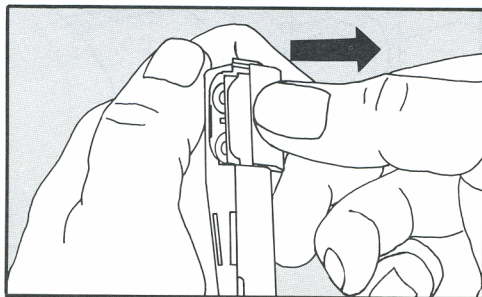
4. Tip the calculator to remove the old batteries.

5. Insert three new batteries. Orient the batteries as shown on the diagram on the back of the calculator. Be certain to observe the polarities (+ and −) as shown.

6. Press the batteries into the compartment using the portion of the battery door that extends beyond the metal contact plate. Press down until the contact plate is lined up with the grooves on the calculator case.
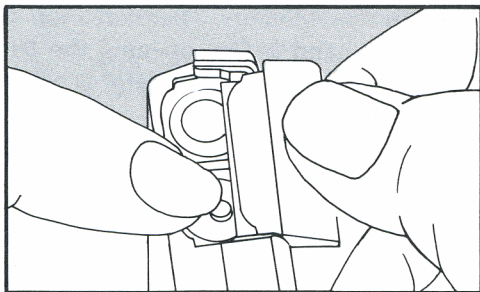
7. Slide the contact plate into the grooves. If necessary, use your finger to push the batteries into the compartment so that the door can slide over them. Slide the door until it latches into place.
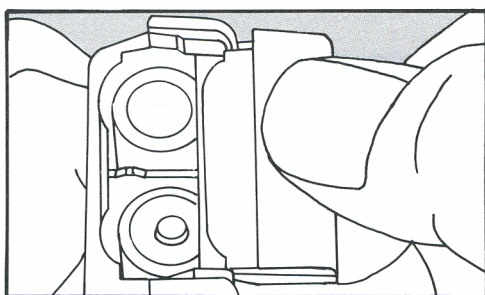
Calculator Maintenance

To clean the display, use a cloth slightly moistened with water. Avoid getting the calculator wet.

Do not lubricate the hinge.

Environmental Limits

In order to maintain product reliability, you should observe the following temperature and humidity limits of the HP-28S:

- 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.

Determining If the Calculator Requires Service

Use these guidelines to determine whether the calculator is functioning properly. If the calculator does require service, see "Limited One-Year Warranty" on page 291 and "If the Calculator Requires Service" on page 293.

If nothing appears in the display when you press [ON]:

1. Check the display contrast.
   a. Press and hold [ON].
   b. Press [+] several times.
   c. Release [ON].
   d. If the display remains blank, press [ON] and repeat steps a, b, and c.

2. Change the batteries, as described on page 286.

3. If steps 1 and 2 don't restore the calculator, it requires service. See "Limited One-Year Warranty" on page 291 and "If the Calculator Requires Service" on page 293.

If the display is visible, but nothing happens when you press keys:

1. Perform a System Halt.

    a. Press and hold $\boxed{\text{ON}}$.

    b. Press $\boxed{\blacktriangle}$.

    c. Release $\boxed{\text{ON}}$.

2. If the calculator is still unresponsive, perform a Memory Reset.

    a. Press and hold $\boxed{\text{ON}}$.

    b. Press and hold $\boxed{\text{INS}}$ and $\boxed{\blacktriangleright}$.

    c. Release $\boxed{\text{INS}}$ and $\boxed{\blacktriangleright}$.

    d. Release $\boxed{\text{ON}}$.

3. If steps 1 and 2 fail to restore the calculator, it requires service. See "Limited One-Year Warranty" on page 291 and "If the Calculator Requires Service" on page 293.

The Repeating Test

If the calculator works, but you think it's not working properly:

1. If you have a printer, turn it on. During the test the calculator prints numbers that are helpful if the calculator requires service.

2. Start the repeating test.

    a. Press and hold $\boxed{\text{ON}}$.

    b. Press $\boxed{\blacktriangleleft}$.

    c. Release $\boxed{\text{ON}}$.

The repeating test proceeds automatically. (If the test doesn't proceed, you probably pressed $\boxed{\text{ON}}\boxed{\blacktriangledown}$ by mistake. This starts a different test, used at the factory, that requires input from the keyboard. Quit this self-test by executing a System Halt, described in step 4 below, and then start the correct repeating test.)

3. Watch for the test message. The test shows horizontal and vertical lines, a blank display, a random pattern, and then it displays the result of the test.

   ■ The message OK-28S indicates that the calculator passed the test.

   ■ A message such as 1 FAIL indicates that the calculator failed the test. The number indicates the nature of the failure. When you send the calculator for service, include the failure number and printed output (if available).

   If you interrupt the repeating test by pressing a key, the test returns a failure message because it didn't expect any keystrokes. *Such a failure message doesn't indicate a problem with the calculator.*

4. Halt the test by performing a System Halt.

   a. Press and hold $\boxed{\text{ON}}$.

   b. Press $\boxed{\blacktriangle}$.

   c. Release $\boxed{\text{ON}}$.

5. If the test returns a failure message, *and you didn't cause the failure by interrupting the test*, the calculator requires service. See "Limited One-Year Warranty" below and "If the Calculator Requires Service" on page 293.

Limited One-Year Warranty

What Is Covered

The calculator (except for the batteries, or damage caused by the batteries) *is 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 with a newer model of equivalent 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.

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 an equivalent or newer model), whether it is under warranty or not. There is a charge for service after the warranty period. Calculators normally are serviced and reshipped within 5 working days of receipt.

Obtaining Service

- **In the United States:** Send the calculator to the Calculator Service Center listed on the inside of the back cover.

- **In Europe:** Contact your HP sales office or dealer or HP's European headquarters 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) 82 81 11

- **In other countries:** Contact your HP sales office or dealer or write to the U.S. Calculator 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 U.S. Calculator Service Center for repair.

    All shipping, reimportation arrangements, and customs costs are your responsibility.

Service Charge

There is a standard repair charge for out-of-warranty service. The Calculator Service Center (listed on the inside of the back cover) can tell you how much this charge is. The full 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 service charges. In these cases, 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. (You must pay the shipping charges for delivery to the service center, whether or not the calculator is under warranty.) Be sure to:

■ Include your return address and 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 the calculator in adequate protective packaging to prevent damage. Such damage is not covered by the warranty, so we recommend that you insure the shipment.

■ Pay the shipping charges for delivery to the Hewlett-Packard service center, whether or not the calculator is under warranty.

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. Refer to the form that was wrapped with the manual. For additional information, contact the Calculator Service Center (see the inside of the back cover).

Regulatory Information

Radio Frequency Interference

U.S.A. The HP-28S generates and uses radio frequency energy and may interfere with radio and television reception. The calculator complies with the limits for a Class B computing device as specified in Subpart J of 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 HP-28S off and on or by removing the batteries), try:

■ Reorienting the receiving antenna.

■ Relocating the calculator 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.

West Germany. The HP-28S and the HP 82240A printer comply with VFG 1046/84, VDE 0871B, and similar non-interference standards.

If you use equipment that is not authorized by Hewlett-Packard, that system configuration has to comply with the requirements of Paragraph 2 of the German Federal Gazette, Order (VFG) 1046/84, dated December 14, 1984.

B

Notes for HP RPN
Calculator Users

Starting with the HP-35 in 1972, Hewlett-Packard has developed a series of handheld scientific and business calculators based upon the RPN stack interface. Although there are many differences in the capabilities and applications of these various calculators, they all share a common implementation of the basic stack interface, which makes it easy for a user accustomed to one calculator to learn to use any of the others.

The HP-28S also uses a stack and RPN logic as the central themes of its user interface. However, the four-level stack and fixed register structure of the previous calculators is inadequate to support the multiple object types and symbolic mathematical capability of the HP-28S. Thus while the HP-28S is a natural evolution of the "original" RPN interface, there are sufficient differences between the HP-28S and its predecessors to require a little "getting used to" if you are accustomed to other RPN calculators. In this appendix, we will highlight the major differences.

The Dynamic Stack

The most dramatic difference in the basic interface of the HP-28S compared with previous HP RPN calculators is the size of the stack. The other calculators feature a fixed, four-level stack consisting of the X-, Y-, Z- and T-registers, augmented by a single LAST X, or L-register. This stack is always "full"—even when you "clear" the stack, all you are doing is filling the stack with zeros.

The HP-28S has no fixed size to its stack. As you enter new objects onto the stack, new levels are dynamically created as they are needed. When you remove objects from the stack, the stack shrinks, even to the point where the stack is empty. Thus the HP-28S can generate a Too Few Arguments error that previous HP RPN calculators could not.

The dynamic versus fixed stack implementation gives rise to the following specific differences between the HP-28S and fixed-stack calculators:

Numbered levels. The indefinite size of the HP-28S stack makes the X Y Z T stack level names inappropriate—instead, the levels are numbered. Thus level 1 is analogous to the X-register, 2 to Y, 3 to Z, and 4 to T. The key labels $1/x$ and x^2 were preserved on the HP-28S for the sake of familiarity—they make the keys more visible than their actual command names INV and SQ, respectively. However, the RPN fixture $X<>Y$ has been renamed SWAP on the HP-28S.

Stack Manipulation. The HP-28S requires a more general set of stack manipulation commands than the fixed-stack calculators. R↑ and R↓, for example, are replaced by ROLL and ROLLD, respectively, each of which require an additional argument to specify how many stack levels to roll. The STACK menu contains several stack manipulation commands that do not exist on the fixed-stack calculators.

No Automatic Replication of the T-register. On fixed-stack calculators, the contents of the T-register are duplicated into the Z-register whenever the stack "drops" (that is, when a number is removed from the stack). This provides a convenient means for constant multiplication—you can fill the stack with copies of a constant, then multiply it by a series of numbers by entering each number, pressing [×], then [CLx] after you have recorded each result. You can't do this on the HP-28S—but it is easy to create a program of the form

$$\ll\ 12345\ *\ \gg\ 'MULT'\ STO$$

where 12345 represents a typical constant. Then all you have to do is press [USER], enter a number and press MULT, enter a new number and press MULT again, and so on, to perform constant multiplication. You can leave successive results on the stack.

Stack Memory. A dynamic stack has the advantage that you can use as many levels as you need for any calculation, without worrying about losing objects "off the top" as you enter new ones. This also has the disadvantage that you can tie up a significant amount of memory with old objects, if you leave them on the stack after you are finished with a calculation. With the HP-28S, you should get in the habit of discarding unneeded objects from the stack.

DROP Versus CLX. In fixed-stack calculators, CLX means "replace the contents of the X-register with 0, and disable stack lift" (see below). Its primary purpose is to throw away an old number, prior to replacing it with a new one—but you can also use it as a means to enter 0. On the HP-28S, CLX is replaced by DROP, which does what its name implies—it drops the object in level 1 from the stack, and the rest of the stack drops down to fill in. No extraneous 0 is entered. Similarly, CLEAR drops all objects from the stack, instead of replacing them with zeros as does its fixed-stack counterpart CLST (CLEAR STACK).

Stack-Lift Disable and ENTER

Certain commands on fixed-stack calculators (ENTER↑, CLX, $\Sigma+$, $\Sigma-$) exhibit a peculiar feature called *stack-lift disable*. That is, after any of these commands is executed, the next number entered onto the stack replaces the current contents of the X-register, rather than pushing it into the Y-register. This feature is entirely absent on the HP-28S. New objects entered onto the stack *always* push the previous stack objects up to higher levels.

The X-register and ENTER on fixed-stack calculators play dual roles that are derived more from the single-line display of the calculators than from the stack structure. The X-register acts as an input register as well as an ordinary stack register—when you key in a number, the digits are created in the X-register, until a non-digit key terminates entry. The ENTER↑ key is provided for separating two consecutive number entries. But in addition to terminating digit entry, the ENTER↑ key also copies the contents of the X-register into Y, and disables stack lift.

On the HP-28S each of these dual roles is separated—there is no stack lift disable. A command line completely distinct from level 1 (the "X-register") is used for command entry. ENTER is used *only* to process the contents of the command line—it does not duplicate the contents of level 1. Note, however, that the [ENTER] key will execute DUP (which copies level 1 into level 2) if no command line is present. This feature of [ENTER] is provided partly for the sake of similarity to previous calculators.

Prefix Versus Postfix

HP-28S commands use a strict postfix syntax. That is, all commands using arguments require that those arguments be present on the stack before the command is executed. This departs from the convention used by previous RPN calculators, in which arguments specifying a register number, a flag number, and so on, are not entered on the stack but are entered *after* the command itself—for example, STO 25, TONE 1, CF 03, and so on. This latter method has the advantage of saving a stack level, but the disadvantage of requiring an inflexible format—STO on the HP-41, for example, must always be followed by a two-digit register number.

Similar operations of the HP-28S are closer in style to *indirect* operations on the fixed-stack calculators, where you can use an *i-register* (or any register, in the case of the HP-41) to specify the register, flag number, and so on, addressed by a command. You can view STO, RCL, and so on, on the HP-28S as using level 1 as an *i-register*. RCL, for example, means "recall the contents of the variable ('register') named in level 1"— equivalent to RCL IND X on the HP-41.

You should be aware also that most HP-28S commands remove their arguments from the stack. If you execute, for example, 123 'X' STO, the 123 and the 'X' disappear from the stack. Without this behavior, the stack would be overloaded with "old" arguments. If you want to keep the 123 on the stack, you should execute 123 DUP 'X' STO.

Registers Versus Variables

Fixed-stack calculators can deal efficiently only with real, floating-point numbers for which the fixed, seven-byte register structure of the stack and numbered data register memory is suitable (the HP-41 introduced a primitive alpha data object constrained to the seven-byte format). The HP-28S replaces numbered data registers with named variables. Variables, in addition to having a flexible structure so that they can accomodate different object types, have names that can help you remember their contents more readily than can register numbers.

If you want to duplicate numbered registers on the HP-28S, you can use a vector:

$$\{ \ 50 \ \} \ 0 \ \text{CON} \ \text{'REG'} \ \text{STO}$$

creates a vector with 50 elements initialized to 0;

$$\ll \ \text{'REG'} \ \text{SWAP} \ \text{GET} \ \gg \ \text{'NRCL'} \ \text{STO}$$

creates a program NRCL that recalls the nth element from the vector, where n is a number in level 1;

$$\ll \ \text{'REG'} \ \text{SWAP} \ \text{ROT} \ \text{PUT} \ \gg \ \text{'NSTO'} \ \text{STO}$$

creates the analogous store program NSTO.

LASTX Versus LAST

The LASTX command on fixed-stack calculators returns the contents of the LASTX (or L) register, which contains the last value used from the X-register. This concept is generalized on the HP-28S to the LAST command, which returns the last one, two, or three arguments taken from the stack by a command (no command uses more than three arguments). Thus 1 2 + LASTX returns 3 and 2 to the stack on a fixed-stack calculator, but 1 2 + LAST returns 3, 1, and 2 to the stack on the HP-28S.

Although the HP-28S LAST is more flexible than its LASTX predecessor, you should keep in mind that more HP-28S commands use arguments from the stack than their fixed-stack calculator counterparts. This means that the LAST arguments are updated more frequently, and even such commands as DROP or ROLL will replace the LAST arguments.

Remember also that UNDO can replace the entire stack, which for simple error recovery may be preferable to LAST.

C

Notes for Algebraic Calculator Users

Many calculators, including the great majority of simple, "four-function" calculators, use variations of the *algebraic* calculator interface. The name derives from the feature that the keystroke sequences used for simple calculations closely parallel the way in which the calculation is specified in algebraic expressions "on paper." That is, to evaluate $1 + 2 - 3$, you press $\boxed{1}\ \boxed{+}\ \boxed{2}\ \boxed{-}\ \boxed{3}\ \boxed{=}$.

This interface works nicely for expressions containing numbers and *operators*—functions like $+$, $-$, $\times$, and $/$ that are written in infix notation between their arguments. More sophisticated calculators allow you to enter parentheses to specify precedence (the order of operations). However, the introduction of prefix functions, like SIN, LOG, and so on, leads to two different variations:

- Ordinary algebraic calculators use a combination of styles—infix operators remain infix, but prefix functions are entered in a postfix style (like RPN calculators). For example, $1 + \text{SIN}(23)$ is entered as $\boxed{1}\ \boxed{+}\ \boxed{2}\ \boxed{3}\ \boxed{\text{SIN}}\ \boxed{=}$. This approach has the advantages of being able to show intermediate results, and of preserving single-key evaluations of prefix functions (that is, without parentheses), but the disadvantage of losing the correspondence with ordinary mathematical notation that is the primary advantage of the algebraic interface.

- "Direct formula entry" calculators, and BASIC language computers that have an immediate-execute mode, allow you to key in an entire expression in its ordinary algebraic form, then compute the result when you press a termination key (variously labeled ⊡ENTER⊡, ⊡ENDLINE⊡, ⊡RETURN⊡, and so on). This approach has the advantage of preserving the correspondence between written expressions and keystrokes, but usually the disadvantage of providing no intermediate results. (The HP-71B CALC mode is an exception.) You have to know the full form of an expression before you start to enter it—it is difficult to "work your way through a problem," varying the calculation according to intermediate results.

Getting Used to the HP-28S

HP-28S operating logic is based on a mathematical logic known as "Polish Notation," developed by the Polish logician Jan Łukasiewicz (*Wookashye'veech*) (1878–1956). Conventional algebraic notation places arithmetic operators *between* the relevant numbers or variables when evaluating algebraic expressions. Łukasiewicz's notation specifies the operators *before* the variables. A variation of this logic specifies the operators *after* the variables—this is termed "Reverse Polish Notation," or "RPN" for short.

The basic idea of RPN is that you enter numbers or other objects into the calculator first, then execute a command that acts on those entries (called "arguments"). The "stack" is just the sequence of objects waiting to be used. Most commands return their results to the stack, where they can then be used as arguments for subsequent operations.

The HP-28S uses an RPN stack interface because it provides the necessary flexibility to support the wide variety of HP-28S mathematical capabilities in a uniform manner. All calculator operations, including those that can not be expressed as algebraic expressions, are performed in the same manner—arguments from the stack, results to the stack.

Nevertheless, using the RPN stack for simple arithmetic is most likely the biggest stumbling block for algebraic calculator users trying to learn to use RPN calculators. RPN is very efficient, but it does require you mentally to rearrange an expression before you can calculate results. But the HP-28S's capability of interpreting algebraic expressions without translation should make the transition from algebraic calculator use more straightforward than has been possible on previous RPN calculators. The four-line display can also help to take away some of the mystery of the stack, by showing you the contents of up to four levels at a time.

For the purpose of evaluating algebraic expressions, the HP-28S is essentially a "direct formula entry" calculator. That is, to evaluate an algebraic expression, all you have to do is precede it with a ⎡'⎤, key in the expression in its algebraic form, including infix operators, prefix functions, and parentheses, and then press ⎡EVAL⎤ to see the result. You can use this method even for simple arithmetic:

⎡'⎤ ⎡1⎤ ⎡+⎤ ⎡2⎤ ⎡−⎤ ⎡3⎤ ⎡EVAL⎤ returns 0.

Except for the preceding ⎡'⎤, these are the same keystrokes you would use on a simple algebraic calculator, where you substitute ⎡EVAL⎤ for ⎡=⎤.

Note   Don't confuse the HP-28S ⎡=⎤ key with that found on algebraic calculators — on the HP-28S, ⎡=⎤ is used for the sole purpose of creating algebraic equations (described in "ALGEBRA" in the Reference Manual).

When you use the HP-28S as a "direct formula entry calculator," each result that you compute is retained on the stack, which takes on the role of a "history stack." This allows you to save old results indefinitely for reuse later. It also allows you to break up large calculations into smaller ones, keeping each partial result on the stack and then combining the results when they are all available. (When carried to the extreme, this is the essence of RPN arithmetic). The stack provides a much easier-to-use and more powerful history stack than the single "result" function available on algebraic or BASIC calculators.

A key feature of the HP-28S is that you really don't need to concern yourself over whether RPN logic is better or worse than algebraic logic. You can choose the logic that is best suited for the problem at hand, and intermix algebraic expressions with RPN manipulations.

D

Menu Map

This appendix shows the commands in each HP-28S menu. The menus are listed in alphabetical order, from ALGEBRA to TRIG. For detailed information about a menu, refer to the Dictionary in the Reference Manual. The Dictionary describes all menus, listed in alphabetical order. For detailed information about a particular command, refer to the Operation Index at the back of the Reference Manual. The Operation Index lists all commands in alphabetical order and gives a page reference to the command's description in the Dictionary.

This appendix doesn't include the menus of the interactive operations offered by CATALOG, FORM, the Solver, and UNITS.

- CATALOG is described in chapter 22 and demonstrated on page 31.

- FORM is described in "Using FORM" on page 112. For details, see "ALGEBRA (FORM)" in the Reference Manual.

- The Solver is described in chapter 8, "The Solver,". For details, see "SOLVE" in the Reference Manual.

- UNITS is described in "The UNITS Catalog" on page 141. For details, see "UNITS" in the Reference Manual.

For each menu in this appendix, the commands are grouped by rows that appear in the display at one time. Pressing ⌈NEXT⌉ moves to the next row, and pressing ▆⌈PREV⌉ moves to the previous row.

The column labeled "Command" is the name that appears in the display. The column labeled "Description" is a short description of the command or its entire name. The column labeled "Page" refers to an example, description, or mention of the command in this manual. For commands without page references, see the Operation Index in the Reference Manual.

ALGEBRA

	Command	Description	Page
Row 1	COLCT	Collect terms	111
	EXPAN	Expand products	111
	SIZE	Size	
	FORM	Form algebraic expression	112
	OBSUB	Object substitute	
	EXSUB	Expression substitute	
NEXT			
Row 2	TAYLR	Taylor series	
	ISOL	Isolate	112
	QUAD	Quadratic form	
	SHOW	Show variable	
	OBGET	Object get	
	EXGET	Expression get	

ARRAY

	Command	Description	Page
Row 1	→ARRY	Stack-to-array	275
	ARRY→	Array-to-stack	274
	PUT	Put element	
	GET	Get element	
	PUTI	Put and increment index	
	GETI	Get and increment index	
NEXT			
Row 2	SIZE	Size	274
	RDM	Redimension	
	TRN	Transpose	264
	CON	Constant array	
	IDN	Identity matrix	
	RSD	Residual	
NEXT			
Row 3	CROSS	Cross product	126
	DOT	Dot product	126
	DET	Determinant	128
	ABS	Absolute value	
	RNRM	Row norm	
	CNRM	Column norm	
NEXT			
Row 4	R→C	Real-to-complex	
	C→R	Complex-to-real	
	RE	Real part	
	IM	Imaginary part	
	CONJ	Conjugate	
	NEG	Negate	

BINARY

	Command	Description	Page
Row 1	DEC	Decimal mode	140
	HEX	Hexadecimal mode	139
	OCT	Octal mode	140
	BIN	Binary mode	140
	STWS	Store wordsize	139
	RCWS	Recall wordsize	
NEXT			
Row 2	RL	Rotate left	
	RR	Rotate right	
	RLB	Rotate left byte	
	RRB	Rotate right byte	
	R→B	Real-to-binary	261
	B→R	Binary-to-real	
NEXT			
Row 3	SL	Shift left	
	SR	Shift right	
	SLB	Shift left byte	
	SRB	Shift right byte	
	ASR	Arithmetic shift right	
NEXT			
Row 4	AND	And	
	OR	Or	
	XOR	Exclusive or	
	NOT	Not	

COMPLEX

	Command	Description	Page
Row 1	R→C	Real-to-complex	83
	C→R	Complex-to-real	83
	RE	Real part	83
	IM	Imaginary part	84
	CONJ	Conjugate	84
	SIGN	Sign	84
NEXT			
Row 2	R→P	Rectangular-to-polar	86
	P→R	Polar-to-rectangular	85
	ABS	Absolute value	85
	NEG	Negate	85
	ARG	Argument	85

LIST

	Command	Description	Page
Row 1	→LIST	Stack-to-list	181
	LIST→	List-to-stack	181
	PUT	Put element	271
	GET	Get element	237
	PUTI	Put and increment index	271
	GETI	Get and increment index	271
NEXT			
Row 2	POS	Position	237
	SUB	Subset	276
	SIZE	Size	271

LOGS

	Command	Description	Page
Row 1	LOG	Common logarithm	78
	ALOG	Common antilogarithm	78
	LN	Natural logarithm	78
	EXP	Exponential	78
	LNP1	Natural log of $1 + x$	78
	EXPM	Exponential minus 1	78
Row 2	SINH	Hyperbolic sine	78
	ASINH	Inverse hyperbolic sine	78
	COSH	Hyperbolic cosine	78
	ACOSH	Inverse hyperbolic cosine	78
	TANH	Hyperbolic tangent	78
	ATANH	Inverse hyperbolic tangent	78

NEXT

MEMORY

	Command	Description	Page
Row 1	MEM	Available memory	188
	MENU	Create custom menu	195
	ORDER	Order variables	184
	PATH	Current path	67
	HOME	Select HOME directory	71
	CRDIR	Create directory	66
NEXT			
Row 2	VARS	Variables in current directory	184
	CLUSR	Clear current directory	184

MODE

	Command	Description	Page
Row 1	STD	Standard number display format	38
	FIX	Fixed number display format	38
	SCI	Scientific number display format	38
	ENG	Engineering number display format	38
	DEG	Degrees angle mode	74
	RAD	Radians angle mode	74
NEXT			
Row 2	CMD	Enables or disables COMMAND	210
	UNDO	Enables or disables UNDO	211
	LAST	Enables or disables LAST	211
	ML	Enables or disables multi-line	208
	RDX,	Enables or disables RDX,	37
	PRMD	Prints and displays modes	

PLOT

	Command	Description	Page
Row 1	STEQ	Store equation	90
	RCEQ	Recall equation	
	PMIN	Plot minima	95
	PMAX	Plot maxima	95
	INDEP	Independent	
	DRAW	Draw	90

NEXT

	Command	Description	Page
Row 2	PPAR	Recall plot parameters	90
	RES	Resolution	
	AXES	Axes	
	CENTR	Center	94
	*W	Multiply width	
	*H	Multiply height	93

NEXT

	Command	Description	Page
Row 3	STOΣ	Store sigma	
	RCLΣ	Recall sigma	
	COLΣ	Sigma columns	
	SCLΣ	Scale sigma	
	DRWΣ	Draw sigma	

NEXT

	Command	Description	Page
Row 4	CLLCD	Clear LCD	
	DGTIZ	Digitize	
	PIXEL	Pixel	
	DRAX	Draw axes	
	CLMF	Clear message flag	
	PRLCD	Print LCD	

PRINT

	Command	Description	Page
	PR1	Print level 1	151
	PRST	Print stack	152
Row 1	PRVAR	Print variable	152
	PRLCD	Print LCD	149
	CR	Carriage right	
	TRAC	Enable or disable Trace mode	150
NEXT			
	PRSTC	Print stack (compact)	
	PRUSR	Print user variables	
Row 2	PRMD	Print modes	

PROGRAM BRANCH

	Command	Description	Page
Row 1	IF	Begin IF clause	226
	IFERR	Begin IF ERROR clause	227
	THEN	Begin THEN clause	226
	ELSE	Begin ELSE clause	226
	END	End program structure	226
NEXT			
Row 2	START	Begin definite loop	228
	FOR	Begin definite loop	229
	NEXT	End definite loop	228
	STEP	End definite loop	230
	IFT	If-Then command	227
	IFTE	If-Then-Else function	226
NEXT			
Row 3	DO	Define indefinite loop	231
	UNTI	Define indefinite loop	231
	END	End program structure	231
	WHIL	Define indefinite loop	232
	REPER	Define indefinite loop	232
	END	End program structure	232

PROGRAM CONTROL

	Command	Description	Page
Row 1	SST	Single step	250
	HALT	Suspend program	234
	ABORT	Abort program	
	KILL	Abort suspended programs	250
	WAIT	Pause program	234
	KEY	Return key string	234
NEXT			
Row 2	BEEP	Beep	234
	CLLCD	Clear LCD	234
	DISP	Display	234
	CLMF	Clear message flag	234
	ERRN	Error number	
	ERRM	Error message	

PROGRAM TEST

	Command	Description	Page
Row 1	SF	Set flag	205
	CF	Clear flag	205
	FS?	Flag set?	225
	FC?	Flag clear?	
	FS?C	Flag set? Clear	
	FC?C	Flag clear? Clear	
NEXT			
Row 2	AND	And	
	OR	Or	
	XOR	Exclusive or	
	NOT	Not	232
	SAME	Same	231
	==	Equal	222
NEXT			
Row 3	STOF	Store flags	156
	RCLF	Recall flags	156
	TYPE	Type	232

REAL

	Command	Description	Page
Row 1	NEG	Negate	78
	FACT	Factorial (gamma)	78
	RAND	Random number	78
	RDZ	Randomize	78
	MAXR	Maximum real	79
	MINR	Minimum real	79
NEXT			
Row 2	ABS	Absolute value	
	SIGN	Sign	
	MANT	Mantissa	
	XPON	Exponent	
NEXT			
Row 3	IP	Integer part	
	FP	Fractional part	
	FLOOR	Floor	272
	CEIL	Ceiling	272
	RND	Round	
NEXT			
Row 4	MAX	Maximum	
	MIN	Minimum	
	MOD	Modulo	
	%T	Percent of total	

SOLVE

	Command	Description	Page
Row 1	STEQ	Store equation	64
	RCEQ	Recall equation	
	SOLVR	Solver variables menu	102
	ISOL	Isolate	110
	QUAD	Quadratic form	108
	SHOW	Show variable	
[NEXT]			
Row 2	ROOT	Rootfinder	

STACK

	Command	Description	Page
Row 1	DUP	Duplicate	178
	OVER	Over	178
	DUP2	Duplicate two objects	178
	DROP2	Drop two objects	179
	ROT	Rotate	178
	LIST→	List-to-stack	181
NEXT			
Row 2	ROLLD	Roll down	178
	PICK	Pick	178
	DUPN	Duplicate *n* objects	178
	DROPN	Drop *n* objects	179
	DEPTH	Depth	181
	→LIST	Stack-to-list	181

STAT

	Command	Description	Page
Row 1	Σ+	Sigma plus	132
	Σ-	Sigma minus	133
	NΣ	Sigma N	134
	CLΣ	Clear sigma	132
	STOΣ	Store sigma	275
	RCLΣ	Recall sigma	264
NEXT			
Row 2	TOT	Total	
	MEAN	Mean	134
	SDEV	Standard deviation	135
	VAR	Variance	135
	MAXΣ	Maximum sigma	
	MINΣ	Minimum sigma	
NEXT			
Row 3	COLΣ	Sigma columns	136
	CORR	Correlation	136
	COV	Covariance	136
	LR	Linear regression	137
	PREDV	Predicted value	137
NEXT			
Row 4	UTPC	Upper chi-square distribution	
	UTPF	Upper Snedecor's f distribution	
	UTPN	Upper normal distribution	
	UTPT	Upper Student's t distribution	
	COMB	Combinations	
	PERM	Permutations	

STORE

	Command	Description	Page
Row 1	STO+	Store plus	
	STO-	Store minus	
	STO*	Store times	
	STO/	Store divide	
	SNEG	Store negate	
	SINV	Store invert	
NEXT			
Row 2	SCONJ	Store conjugate	

STRING

	Command	Description	Page
Row 1	→STR	Object-to-string	258
	STR→	String-to-object	175
	CHR	Character	156
	NUM	Character number	156
	→LCD	String-to-LCD	157
	LCD→	LCD-to-string	157
NEXT			
Row 2	POS	Position	
	SUB	Subset	
	SIZE	Size	258
	DISP	Display	156

TRIG

	Command	Description	Page
Row 1	SIN	Sine	74
	ASIN	Arc sine	74
	COS	Cosine	74
	ACOS	Arc cosine	74
	TAN	Tangent	74
	ATAN	Arc tangent	74
NEXT			
Row 2	P→R	Polar-to-rectangular	76
	R→P	Rectangular-to-polar	76
	R→C	Real-to-complex	76
	C→R	Complex-to-real	76
	ARG	Argument	76
NEXT			
Row 3	→HMS	Decimal to hours-minutes-seconds	76
	HMS→	Hours-minutes-seconds to decimal	76
	HMS+	Hours-minutes-seconds plus	76
	HMS−	Hours-minutes-seconds minus	76
	D→R	Degrees-to-radians	77
	R→D	Radians-to-degrees	77

Key Index

This index describes the actions of the keys on the calculator keyboard. First is an alphabetical index of the keys on the left-hand keyboard, followed by an alphabetical index of the keys on the right-hand keyboard. Last is an index of the keys on the cursor menu (the white labels above the top row of the right-hand keyboard).

This index includes shifted keys such as ■ ARRAY and ■ OFF . It doesn't include character keys such as A through Z and 0 through 9 , which always write a character in the command line. (Other character keys include delimiters such as (, operators such as =, and symbolic constants such as ■ π . These characters have special meaning to the calculator, but their keys are simply character keys.) If you don't find a key listed in this index, it is a character key.

For each key, there is a brief description of its action and a page reference. If the key isn't mentioned in this manual, or for additional information about any key, look in the Operation Index at the back of the Reference Manual.

Left-hand Keyboard

Key	Description	Page
■ ALGBRA	Selects the ALGEBRA menu.	110
■ ARRAY	Selects the ARRAY menu.	124
■ BINARY	Selects the BINARY menu.	138
■ BRANCH	Selects the PROGRAM BRANCH menu.	222
■ CATALOG	Starts the command catalog.	196
■ COMPLX	Selects the COMPLEX menu.	83
■ CONTRL	Selects the PROGRAM CONTROL menu.	234
LC	Switches lower-case mode on or off.	168
■ LIST	Selects the LIST menu.	102
■ MENUS	Switches Menu Lock on or off.	192
■ MEMORY	Selects the MEMORY menu.	182
■ PRINT	Selects the PRINT menu.	149
■ REAL	Selects the REAL menu.	78
■ STACK	Selects the STACK menu.	176
■ STAT	Selects the STAT menu.	131
■ STORE	Selects the STORE menu.	191
■ STRING	Selects the STRING menu.	156
■ TEST	Selects the PROGRAM TEST menu.	225
■ UNITS	Selects the UNITS catalog.	141
α	Switches entry mode.	171

Right-hand Keyboard

Key	Description	Page
ATTN (ON)	Aborts program execution; clears the command line; exits catalogs, FORM, plot displays.	216
CHS	Changes the sign of a number in the command line or executes NEG.	168
■ CLEAR	Clears the stack.	179
■ COMMAND	Moves an entry from the command stack to the command line.	174
■ CONT	Continues a halted program.	235
■ CONVERT	Performs a unit conversion.	143
■ CUSTOM	Seclects the last-displayed custom menu.	192
■ d/dx	Derivative.	117
DROP	Drops one object from the stack.	179
■ EDIT	Copies the object in level 1 to the command line for editing.	173
EEX	Enters exponent in command line.	168
ENTER	Parses and evaluates the command line.	173
EVAL	Evaluates an object.	118
■ LAST	Returns last arguments.	179
■ LOGS	Selects the LOGS menu.	77
■ MODE	Selects the MODE menu.	36
NEXT	Displays the next row of menu labels.	192
ON (ATTN)	Turns the calculator on; aborts program execution; clears the command line; exits catalogs, FORM, plot displays.	216
■ OFF	Turns the calculator off.	20
■ PLOT	Selects the PLOT menu.	89
■ PREV	Displays the previous row of menu labels.	192

Key	Description	Page
■ PURGE	Purges one or more variables.	183
■ RCL	Recalls the contents of a variable, unevaluated.	183
■ ROLL	Moves the level $n+1$ object to level 1.	178
SOLV	Selects the SOLVE menu.	99
STO	Stores an object in a variable.	183
■ SWAP	Swaps the objects in levels 1 and 2.	178
TRIG	Selects the TRIG menu.	74
■ UNDO	Replaces the stack contents.	180
USER	Selects the USER menu.	49
■ VIEW↑	Moves the display window up one line.	177
■ VIEW↓	Moves the display window down one line.	177
■ VISIT	Copies an object to the command line for editing.	173
[']	Delimiter for names and symbolic expressions.	51
■ x^2	Squares a number or matrix.	40
■ $1/x$	Inverse (reciprocal).	40
+	Adds two objects.	41
−	Subtracts two objects.	41
×	Multiplies two objects.	41
÷	Divides two objects.	42
■ %	Percent.	43
■ %CH	Percent change.	43
■ ^	Raises a number to a power.	42
■ $\sqrt{x}$	Takes the square root.	40
■ $\int$	Definite or indefinite integral.	120
■	Shift key.	29
⬖	Selects cursor menu or restores last menu.	168
◀	Backspace.	168
■ →NUM	Forces a numerical result.	75

Cursor Menu

The cursor menu is labeled in white above the menu keys (the top row of the right-hand keyboard). The cursor menu is active when the command line is present and no menu labels are displayed. To select the cursor menu when menu labels are displayed, press [↔]. To restore the previous menu, press [↔] a second time.

Key	Description	Page
[INS]	Switches between Replace and Insert modes.	167
■ [INS]	Deletes all characters to the left of the cursor.	168
[DEL]	Deletes character at cursor.	167
■ [DEL]	Deletes character at cursor and all characters to the right.	168
[▲]	Moves cursor up.	167
■ [▲]	Moves cursor up all the way.	168
[▼]	Moves cursor down.	167
■ [▼]	Moves cursor down all the way.	168
[◄]	Moves cursor left.	167
■ [◄]	Moves cursor left all the way.	168
[►]	Moves cursor right.	167
■ [►]	Moves cursor right all the way.	168

Subject Index

Page numbers in **bold** type indicate primary references; page numbers in regular type indicate secondary references.

Local variables, 80, 86, 147, 179, 222–223, 242, 259
  evaluation of, 254
  nested, 270
Logarithmic functions, 77–78
Loop structures, 228
Low memory, 188–190
Lowercase mode, 26, 28

M

Maintenance, 289
Mantissa, 38
Matrix, defined, 124
Matrix operations, 263
Maximum, of a expression, 100
Mean, 134
Median, defined, 272
MEDIAN program, 273–275
Memory, low, 188–190
Memory Reset, 20, 217
Menu keys, 27, 31
Menu labels, 27, 31
Menu Lock, 169
Merging terms, 115
Message, printing, 151
Miles per hour conversion, 145
Millimeter conversion, 144
Minimum, of a expression, 100
Modes, 205–214
  indicated by cursor, 172
Moving stack objects, 178
MULTI program, 253–255
Multi-line mode, 208

N

Name-class objects, 199–201
Names, **159–160**
  quoted and unquoted, 57
Negation, 40, 79
Negative number, 39
Nested program structures, 233
  definite loops, 270–271
  local variable structures, 270
  user functions, 245

Newline character, 169
Number display mode, 37, 209
Numerical integration, 122–123
Numerical result mode, 203–204
Numerical variable, 49

O

Object classes, 199
Object types, 26–29
Objects, 154
Off, automatic, 20
One-number functions, 40
Opening the case, 18
Operation, 164–165
Order of arguments, 41, 43
Ounce conversion, 147–148
Overflow, 212
O→G program, 147

P

PAD program, 257–258
Parent directory, 60, 183, 275
Percentages, 43
Performance, maximizing, 190–191
Period, 169
Pi, 74–75
Plotting, **89–97**
Plotting parameters, 89
Polar coordinates, 84–88
Postfix notation, 25
Powers, 42
Predicted values, 137
Prefixed units, 144
PRESERVE program, 258–259
Principal value, 206
Printer port, location, 19
Printing a plot, 91
Procedure-class objects, 199, 201–204
Program structures, 161
  evaluating, 201

Programs, 160–161
  in algebraics, 263
  as arguments, 254
  evaluating, 201–202
Proposition, 162
PSUM program, 86–88
Purging
  a directory, 187
  a variable, 52

Q

Quadratic expressions and equations, 107
Quoted names, 57

R

Radians angle mode, 73
Radix mark, defined, 36
Random numbers, 78
Real numbers, 155
Recalling a variable, 50, 56
Reciprocal, 40
Recovery modes, 210–211
Recursion, 246–247, 249
RENAME program, 54–55
Renaming a variable, 52
Repeating test, 218
Reserved names, 159–160
Resetting memory, 20
Restoring the stack, 180
Results mode, 207
Root of an equation, 107
Roots, 42
RPN, 25
Running record, printing, 150

S

Scale of a plot, 91
Self-tests, 218–219
Separators, 169
Service, 293–295
Shift key, 27, **29**

ΣGET program, 265
ΣX2 program, 266
ΣXY program, 267
ΣY2 program, 266
Single-step execution, 250–253
Solver, 63–64, **98–109**
SORT program, 270–272
Spacing of printed output, 214
Speed of printing, 213
Square, 40
Square root, 40
Stack, **176–181**
Stack, 22, 272
  clearing, 44
  printing, 152
Stack diagram, defined, 240
Stack flags, 225
Stack levels, 27, 31
Stack logic, 25
Standard deviation, 135
Statistics parameters, 136, 263
Status, preserving, 258
Storing plot parameters, 96
Strings, 156–157, 258
Structured programming, 202, 241, 255
Subdirectory, 60, 183, 275
Subprograms, 260
SUMS program, 263–264
Symbolic constants, 163
Symbolic integration, 121
Symbolic result mode, 203
System Halt, 217
System of linear equations, 130

T

Taylor series, 120
Temperature conversion, 143–144
Test functions and commands, 224
Time value of money, 103–106
Trace printing, 150, 213
Translating a plot, 93
Transpose, 264
Trigonometric functions, 73–77
Two-number functions, 40
Types of objects, 26, 28

U

Underflow, 212
Unit catalog, 26, 29
Unit strings, 144–145
Unquoted names, 57
Usage of commands, 197
User flags, 225
User functions, 79–81, 161, 202, 242
 nested, 245
User memory, 48
User menu, custom, 235

V

Variables, 48
 creating, 49, 54
 isolating, 109–116
 printing, 152
 purging, 52
Variance, 135
Vectors, defined, 124

W

Warranty, 291–293
Wordsize, 138–139

X, Y, Z

Zero of an expression, 92, **98–100**,
 107, 162

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, refer to 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 Calculator Service Center:

Hewlett-Packard
Calculator Service Center
1030 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.
(503) 757-2002

If you are outside the United States, refer to appendix A for information on locating the nearest service center.