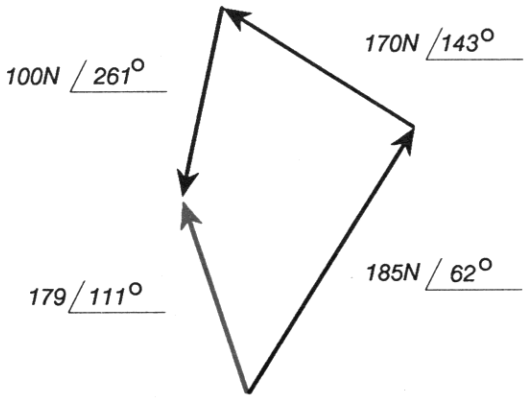
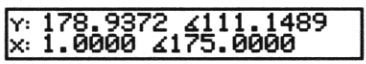


Now calculate the 175° component of this result.

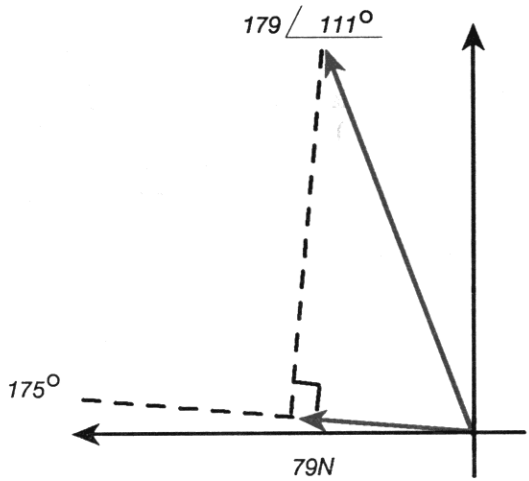
1 ENTER 175 ■ COMPLEX



■ MATRIX ▼ DOT

Thus, the resulting sum has a component of approximately 79 Newtons in the direction of 175°.

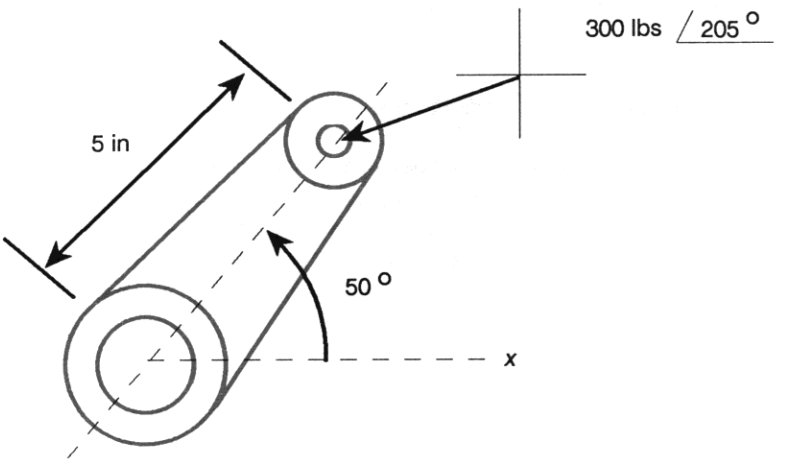
EXIT



Example: Computing Moments. To compute the moment of two vectors, use the CROSS (*cross product*) function. The cross product of two vectors is a third orthogonal vector. However, when two complex numbers are crossed, the HP-42S simply returns a real number that is equal to the signed magnitude of the resulting moment vector.

Find the moment generated by the force acting through the lever in the illustration below, where

$$\vec{M} = \vec{r} \times \vec{F}.$$



Select Degrees and Polar modes. (You can skip this step if you have already selected these modes.)

[■] [MODES] [DEG] [■] [MODES] [POLAR]

```
Y: 67.0820 ∠63.4349
X: 78.8586
```

Key in the radius vector and the force vector.

5 [ENTER] 50 [■] [COMPLEX]

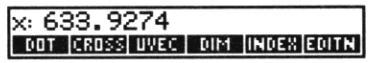
```
Y: 78.8586
X: 5.0000 ∠50.0000
```

300 [ENTER] 205 [■] [COMPLEX]

```
Y: 5.0000 ∠50.0000
X: 300.0000 ∠-155.0000
```

Calculate the cross product.

■ MATRIX ▼ CROSS



The moment vector has a magnitude of 634 and, since the result is positive, the vector points up, perpendicular to the plane of this page.*

EXIT

Storing Complex Numbers

Complex-Number Variables

When you store a complex number into a variable, the variable name is added to the complex-variable catalog. To display a catalog menu containing all of the complex-number variables, press ■ CATALOG CPX . To recall a variable from the catalog, press the corresponding menu key. Refer to chapter 3 for details on using variables and catalogs.

Making the Storage Registers Complex

Normally, each storage register can hold only a real number or an Alpha string. However, you can change the type of the *REGS* matrix to complex so that each storage register holds a complex number.

* If the problem you're working requires a true (three dimensional) vector as a result, use a 1 × 3 matrix to represent each vector in three dimensions.

To make the storage registers complex:

1. Enter zero as a complex number: 0 [ENTER] [■] [COMPLEX].
2. Press [STO] [+] [REGS] to add the complex number (zero) to the *REGS* matrix.

Since the result of any arithmetic is complex if either operand is complex, this procedure makes the storage registers complex. The procedure will fail if any of the storage registers contain an Alpha string.

To make the storage registers real:

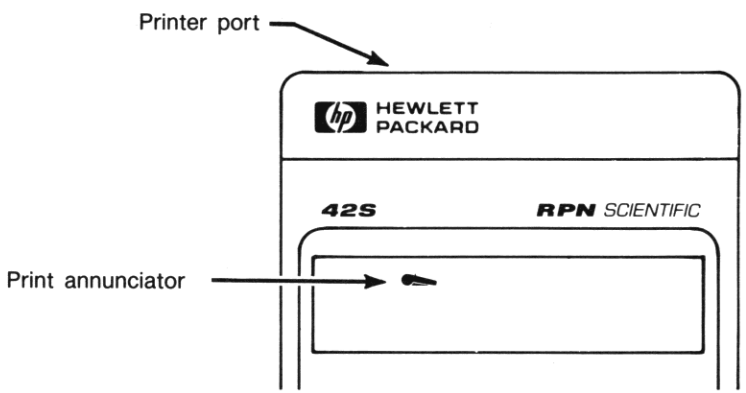
1. Press [RCL] [REGS] to recall a copy of the storage registers into the X-register.
2. Press [■] [COMPLEX] to separate the complex matrix into two real matrices.
3. Press [x≷y] to move the matrix of real-parts into the X-register.
4. Press [STO] [REGS].

7

Printing

The HP-42S prints information using the HP 82240A Infrared Printer, which accepts the infrared signal generated by the calculator's printer port.

The print annunciator ( ) comes on whenever the calculator sends information through its printer port.

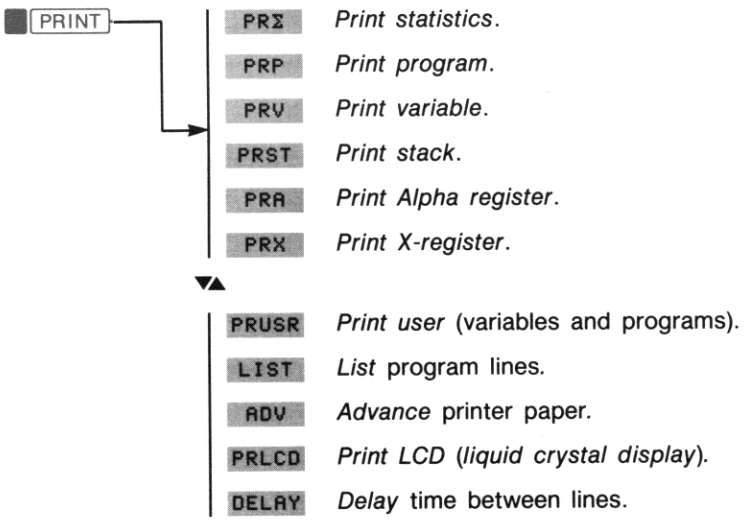


With a printer you can:

- Print intermediate and final results, including all types of data.
- Keep a running record of your keystrokes and calculations.
- List the names of programs and variables stored in the calculator.
- Print complete and partial program listings.
- Print a copy of the display.

Common Printing Operations

The first two rows of the PRINT menu contains these print functions:



Here are a few common printing tasks:

To enable printing: Press PRINT ▲ PON (*printing on*). The PRON function sets flags 21 (printer enable) and 55 (printer existence).

The infrared printer port remains enabled until you disable it by pressing PRINT ▲ POFF (*printing off*). The PROFF function clears flags 21 and 55.

To print the contents of the X-register: Press PRINT PRX.

To print the contents of a variable:

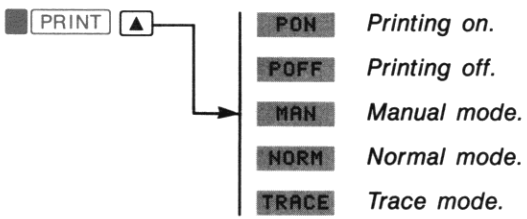
1. Press PRINT PRV.
2. Select the variable from the catalog, or type the variable name using the ALPHA menu.

For example, to print the contents of the storage registers (which are stored as a matrix named *REGS*), press ■ PRINT PRV REGS.

To print the contents of the Alpha register: Press ■ PRINT PRA.

Printing Modes

How and *when* information is sent to the printer depends on the current modes. The functions for controlling the printing modes are in the third row of the PRINT menu:



To select a printing mode:

1. Press ■ PRINT ▲.
2. Press one of the following:
   - MAN (*Manual* mode). Use this mode when you want the calculator to print only when a print function is executed. The VIEW and AVIEW functions can also generate printer output in this mode. (This is the default mode.)
   - NORM (*Normal* mode). Use this mode when you want to print a record of prompts and keystrokes.
   - TRACE (*Trace* mode). Use this mode when you want to print a record of prompts, keystrokes, and results. If a program is running, each instruction is printed as it executes. This mode is primarily intended for testing and debugging programs.

Flags That Affect Printing

There are several flags that affect how and when information is printed. For example, to cause all printing to be double-wide, set flag 12 ([FLAGS] SF 12). To return to normal-width printing, clear flag 12 ([FLAGS] CF 12).

Flag(s)	Purpose	Page(s)
12	Double-wide printing.	274
13	Lowercase printing.	274
15 and 16	Printing mode.	274
21 and 55	Printer enable and printer existence.	131 and 132

Printing Speed and Delay Time

Since the HP-42S is capable of sending information faster than it can be printed by the HP 82240A Infrared Printer, the calculator uses a *delay* time to avoid losing information. To optimize printing speed, set the delay time slightly greater than the time it takes for your printer to print a single line of information.

To set the printing delay time:

1. Key the delay time into the X-register (in seconds). The longest delay time you can set is 1.9 seconds.
2. Press [PRINT] [▼] DELAY.

If you're operating the printer without an AC adapter, printing speed will slow down as the batteries discharge. If you are using the longest delay time (1.9 seconds) and your printer is still too slow, replace the batteries or connect an AC adapter. Operating the printer with batteries this low (without an AC adapter) will likely result in poor infrared communication and may damage the printer.

Low Calculator Batteries

To conserve battery power, the HP-42S will not transmit data to the printer when the  annunciator is on. If low battery power occurs after you've started printing, printing stops and the calculator displays `Batt Too Low To Print`. The calculator automatically returns to Manual mode.

Calculator Functions That Print

If printing is enabled (after executing PRON), the VIEW and AVIEW functions automatically generate printed output (in addition to performing their normal functions).

For more information on how these functions and flags 21 and 55 affect program execution, refer to chapter 9, "Program Input and Output."

Printing Graphics in the Display

The PRLCD ([PRINT] [▼] **PRLCD**) function copies the display to the printer, pixel for pixel. The primary purpose for this function is to print graphics that you create in the display using the PIXEL and AGRAPH functions (page 135).

The "PLOT" program on page 160 creates a plot in the display and then uses the PRLCD function to print it.

Printing Programs

To print an entire program:

1. Press [PRINT] **PRP** (*print program*).
2. Select the program from the catalog, or type the program name (global label) using the ALPHA menu.

The PRP function prints the entire program, even if the global label you specify is not the first line of the program.

If you do not specify any label (■ PRINT PRP ENTER ENTER), the calculator prints the current program.

To print part of a program:

1. Position the program pointer to the line where you want to start the listing (page 111).
2. Press ■ PRINT ▼ LIST .
3. Key in the number of lines you want to print, *nnnn*. (If you enter fewer than four digits, complete the instruction by pressing ENTER .)

The program listing begins with the current line and continues for *nnnn* lines or until an END instruction is encountered.

Character Sets

Some characters are not printed as you see them in the display. This is because the character set used in the HP-42S does not directly match the character set used in the HP 82240A Infrared Printer. Compare the character table in appendix E of this manual with the character set listed in the HP 82240A owner's manual.

Part 2

Programming

Page 108 8: Simple Programming

121 9: Program Input and Output

141 10: Programming Techniques

166 11: Using HP-41 Programs

8

Simple Programming

Part 1 of this manual introduced you to several functions and operations that you can use *manually* (from the keyboard). In this chapter you will learn how *programs* can be used to store and execute a sequence of functions. More specifically, you will learn:

- How to key a program into memory.
- How to edit (change) a program.
- How to execute (run) a program.
- What happens when an error causes a program to stop.
- About the parts of a program.
- How to clear a program from memory.

The programming information in this manual (chapters 8, 9, and 10) should give you a good start at writing your own programs. If you're interested in more advanced programming information and techniques, refer to the *HP-42S Programming Examples and Techniques* manual (part number 00042-90020).

An Introduction to Keystroke Programming

The sequence of steps a program uses to perform a calculation are the same as the steps you would execute when solving the problem manually. By programming your calculator you can repeat operations or calculations without repeating the keystrokes every time.

For example, consider the formula for the area of a circle:

$$A = \pi r^2.$$

To calculate the area of a circle with a radius of 5, you would key in the radius, square it, and then multiply by π.

5 [f] [x^2] [f] [π] [×]

```
Y: 0.0000
X: 78.5398
```

The keystrokes [f] [x^2] [f] [π] [×] can be stored as a program and then executed any number of times for circles of different radii. Such a program might look like this:

```
01 LBL "AREA"
02 X↑2
03 PI
04 ×
05 END
```

This program assumes that the radius is in the X-register when the program runs. To calculate an area, you would key in a radius and then run the program. The result (the area of the circle) is left in the X-register when the program ends.

The label (line 01) identifies the program so you can refer to it by name. The END instruction (line 05) separates this program from the next program in memory.

Example: Keying In and Running a Program. To key a program into the calculator, press [f] [GTO] [·] [·] to move to a *new program space* (page 118), and then press [f] [PRGM] to select *Program-entry mode*.

[f] [GTO] [·] [·] [f] [PRGM]

```
00▶{ 0-Byte Prgm }
01 .END.
```

Key in the program listed above.

[f] [PGM.FCN] [LBL] AREA [ENTER]

```
00 { 8-Byte Prgm }
01▶LBL "AREA"
```

The next three lines are the *body* of the program—that is, the part that calculates the area of the circle. As you press the keys, the calculator records them and automatically numbers them as program steps.

■ x^2

```
01 LBL "AREA"
02▶X↑2
```

■ π

```
02 X↑2
03▶PI
```

[×]

```
03 PI
04▶×
```

The calculator has automatically provided an END statement for you, so the program is complete. If you want to review the program before exiting Program-entry mode, use [▼] and [▲] to move up and down through the lines of the program.

[EXIT]

```
Y: 0.0000
X: 78.5398
```

Now you can use the program to calculate the area of any circle given the radius, *r*. Key in a radius of 5 and run the program.

5 [XEQ] AREA

```
Y: 78.5398
X: 78.5398
```

The result is the same as when you solved the problem manually.

Find the area of a circle with a radius of 2.5.

2.5 [XEQ] AREA

```
Y: 78.5398
X: 19.6350
```

Divide the two results.

[÷]

```
Y: 78.5398
X: 4.0000
```

The area of a circle with a radius of 5 is 4 times greater than the area of a circle with a radius of 2.5.

Program-Entry Mode

The [■] [PRGM] key toggles the calculator in and out of Program-entry mode. In Program-entry mode, functions and numbers you key in are saved as program instructions.

The Program Pointer

While working the example above, you may have noticed the ▶ character in the display. This is the *program pointer*. It points to the *current program line*. If the current program line is too long for the display, press and hold the [■] [SHOW] key to display the entire line.

Moving the Program Pointer

The following instructions are nonprogrammable, so you can execute them in or out of Program-entry mode to move the program pointer.

To move the program pointer to:	**Press:**
The next program line.	[■] [SST] (or [▼] if no menu is displayed)
The previous program line.	[■] [BST] (or [▲] if no menu is displayed)
Line number *nnnn* of the current program.	[■] [GTO] [.] *nnnn*
A global label.	[■] [GTO] [.] [ENTER] *label* [ENTER]
A new program space.	[■] [GTO] [.] [.]

Inserting Program Lines

Instructions keyed into a program are inserted immediately *after* the current program line and the pointer advances to the new line. Therefore, to insert a program line between lines 04 and 05 of a program, you would move the pointer to line 04 and then key in the instruction.

Deleting Program Lines

To delete a program line, position the program pointer to the line you want to delete, and then press [◀]. When you delete a line, the program pointer moves to the previous line.

To delete several consecutive program lines, use the DEL (*delete*) function (page 120).

Executing Programs

In general, there are two ways to run programs:

- *Normal execution.* Program instructions continue to execute until an instruction that stops program execution is encountered (such as STOP, PROMPT, RTN, or END) or until you manually stop the program by pressing [R/S] or [EXIT].
- *Stepwise execution.* Program instructions are executed one at a time as you *step* through the program with the ■ [SST] key. This method of executing a program is especially useful when you are *debugging* a program (testing for errors).

Remember, Program-entry mode must be *off* to run a program.

Normal Execution

To execute a program using the program catalog:

1. Press [XEQ] or ■ [CATALOG] PGM.
2. Press the menu key corresponding to the program you want to execute.

This method is used in the example on page 110.

To assign a program to the CUSTOM menu:

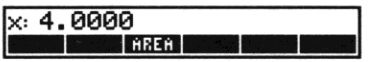
1. Press ■ [ASSIGN] PGM.
2. Press the menu key corresponding to the program you want to assign.

3. The CUSTOM menu has three rows; use ▼ or ▲ to display the row you want and then press the menu key where you want the program assigned.

For example, assign the "AREA" program to the CUSTOM menu.

■ ASSIGN PGM AREA

√x (*the third menu key*)



Now, each time you want to calculate the area of a circle, key in the radius and then press AREA.

5 AREA

3.25 AREA

EXIT

Running a Program With R/S

To run the current program beginning with the current program line, press R/S *(run/stop)*. If you hold the R/S key down, the calculator displays the current program line (▶); that is, the line to be executed next. If you hold R/S down until NULL appears, the program does not begin running when you release the key.

You can position the program pointer to the top of the current program by executing the RTN function when Program-entry mode is off. Therefore, to run the current program (beginning with the first line), press ■ PGM.FCN RTN and then R/S.

Stopping a Program

To stop a running program press [R/S] or [EXIT]. Execution halts after the current instruction is completed. To resume execution, press [R/S] again.

Testing and Debugging a Program

The HP-42S allows you to execute any program one step at a time with the ■[SST] key. This feature is particularly useful when you are trying to find a *bug* (error) in a program or when you just want to see how each instruction in a program works. (Note that if there is no menu displayed, the [▼] and [▲] keys can be used to execute the ■[SST] and ■[BST] functions.)

While testing or debugging a program, you may want to use Trace mode to print a running record of each program step as it is executed. To select Trace mode, press ■[PRINT] [▲] TRACE.

To execute a program one step at a time:

1. Position the program pointer to the label or line number where you want to start executing the program. If you skip this step, execution will begin with the current program line.
2. Be sure Program-entry mode is *off*. If any data is needed at the start of the program, enter it.
3. Press and hold ■[SST] to display the current program line. When you release ■[SST], the instruction is immediately executed and the program pointer advances.

   If you hold the ■[SST] key down too long, NULL appears and the program instruction is *not* executed when you release the key.

When the program pointer reaches the end of the current program, it *wraps* around to the first line.

You can move the program pointer up (backwards) through a program with the ■[BST] key. The ■[BST] key:

- Moves the program pointer *without* executing program instructions.

- Repeats when you hold the key down.

Error Stops

If an error occurs while a program is running, program execution halts, and the appropriate error message is displayed. The error message disappears when you press a key.

The program pointer stops at the line that generated the error. To view the line, select Program-entry mode (■ PRGM).

A running program will ignore an error if flag 24 (*range ignore*) or flag 25 (*error ignore*) is set. Refer to appendix C for more information on these flags.

The Basic Parts of a Program

Program Lines and Program Memory

As you've already seen, when the HP-42S is in Program-entry mode, keystrokes you enter are not immediately executed, but are stored in program memory as instructions. Each instruction occupies a single program line, which is automatically numbered.

Types of Program Lines. Program lines are divided into several categories. A program line may contain:

- A program label (such as LBL "AREA").
- A complete instruction (such as a simple numeric function, like +, or an instruction that includes a parameter, like STO 14).
- A complete number (called a *numeric constant*).
- An Alpha string of up to 15 characters (called an *Alpha constant*).

Memory Requirements. Program steps may vary in size from 1 to 16 bytes. At the top of each program (line 00) the calculator displays the size of the current program in bytes.

If you run out of memory while trying to enter a program line, the calculator displays `Insufficient Memory`. Refer to appendix B, "Managing Calculator Memory."

Program Labels

A label is an identifier placed at the beginning of a series of program steps. Program labels may be used anywhere in a program. Generally, a program starts with a *global label*. Within a program, individual routines can be identified with *local labels*.

Global Labels. Global labels use Alpha characters and are distinguished by quotation marks around the label name.* For example, the program at the beginning of this chapter has a global label:

```
01 LBL "AREA"
```

Global labels may be one to seven characters long. The single-letter names `A` through `J` and `a` through `e` are reserved for local Alpha labels (which are displayed *without* quotation marks).

Global labels:

- Can be accessed no matter where the program pointer is located.
- Are listed in the program catalog ([CATALOG] [PGM]).
- Can be assigned to the CUSTOM menu.
- Should be unique within calculator memory to avoid confusing one program with another.

Local Labels. There are two types of local labels: *numeric* and *Alpha*.

- Numeric labels are identified by two digits, `LBL 00` through `LBL 99`. (`LBL 00` through `LBL 14` are called *short form* local labels because they use less memory.)
- Local Alpha labels use single Alpha characters, `LBL A` through `LBL J` and `LBL a` through `LBL e`.

* To key in a global label that begins with a digit character, select an ALPHA submenu and then type the digit. This forces the digit to become an Alpha character. For example, to key in LBL "1", press [PGM.FCN] [LBL] [ABCDE] 1 [ENTER]. Without [ABCDE], the label is interpreted as LBL 01.

Local labels are used to mark and provide access to various segments of a program. The primary purpose for local labels is to facilitate program *branching*. Refer to "Branching" in chapter 10.

Local labels can be:

- Accessed only within the current program.
- Duplicated in separate programs. That is, local labels do not need to be unique within calculator memory, but they should be unique within each program. (It is possible to use duplicate local labels within a single program if you consider the search patterns used to find local labels. Refer to page 148.)

The Body of a Program

The body of a program is where all the work is done. For example, the body of the "AREA" program is:

```
02 X↑2
03 PI
04 ×
```

This program contains two functions (X↑2 and ×) and a numeric constant (PI).

Constants

Numeric Constants. A numeric constant is simply a number in a program. When the line is executed the number is placed in the X-register, lifting the stack just as if you keyed the number in from the keyboard.

The PI function ([■] [π]) operates like a numeric constant. Thus, the "AREA" program would return exactly the same result if line 03 looked like this:*

```
03 3.14159265359
```

* Although the program would run the same, keying in the 12-digit approximation for π takes 14 bytes of program memory; the PI function requires only 1 byte.

Consecutive Numeric Constants. Because numeric constants in programs are on different program lines, [ENTER] is not needed to separate them. Consider these two programs:

```
01 12                    01 12
02 ENTER                 02 17
03 17                    03 ×
04 ×
```

Both programs produce the same result (12 × 17), however the one on the right is one line shorter and saves one byte of program memory. To key in the program lines on the right, press 12 [ENTER] [◀] 17 [×].

Program ENDs

Programs are separated from one another with END instructions. The last program in memory uses the *permanent* END, which appears in the display as `.END.`.

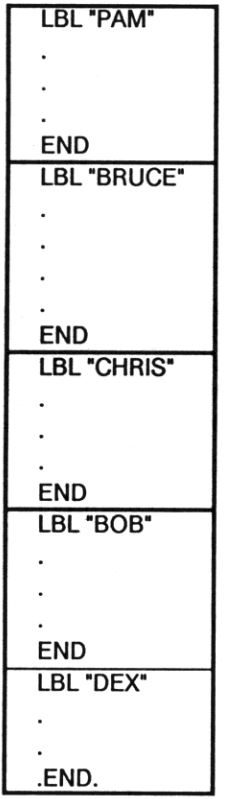
After the first program in memory, you should insert an END between subsequent programs so they will be considered as separate programs, and not just labeled routines within the same program. There are two ways to enter an END at the end of a program:

- Press [■] [GTO] [·] [·]. This procedure automatically inserts an END after the last program in memory and positions the program pointer to the new program space at the bottom of program memory. This space contains the *null* program:

  ```
  00 { 0-Byte Prgm }
  01 .END.
  ```

- *Or,* manually execute the END function (press [XEQ] [ENTER] END [ENTER] or use the function catalog).

Because END instructions separate programs, deleting an END causes the two programs to be joined as one program. You cannot delete the permanent `.END.`.



Clearing Programs

To clear an entire program from memory:

1. Press ■ CLEAR CLP .
2. Specify the program you want to clear using *one* of the following:
   - Press the menu key corresponding to a global label in the program.
   - Use the ALPHA menu to type a global label (ENTER *label* ENTER).
   - *Or,* press ENTER ENTER to clear the *current program.*

To clear a portion of a program:

1. Press ■ PRGM to select Program-entry mode (if the calculator is not already in Program-entry mode).
2. Position the program pointer to the first line in the range of lines you want to delete.
3. Press ■ CLEAR ▼ DEL (*delete*).
4. Key in the number of lines you want to delete.

For example, to delete lines 14 through 22 of the current program, you would press: ■ PRGM (to select Program-entry mode), ■ GTO · 14 ENTER (to position the program pointer to line 14), ■ CLEAR ▼ DEL 9 ENTER (to delete 9 program lines).

The DEL function deletes program lines only if the calculator is in Program-entry mode.

9

Program Input and Output

An *interactive* program has two general characteristics:

- *Input.* The program prompts you to key in information or make a selection.
- *Output.* The program presents results in a meaningful format using the display or a printer.

This chapter covers functions and techniques for making programs easier to use. You'll learn about:

- Prompting for values and using variable menus.
- Displaying labeled output and messages.
- Printing during program execution.
- Working with Alpha data.
- Displaying graphics.

Using the INPUT Function

Using the INPUT function is one of the simplest ways for a program to prompt for data to be stored into a variable or register. When an INPUT instruction is executed:

- The current value of the variable or register is recalled into the X-register. If you use a new variable name, the INPUT function automatically creates the variable and assigns an initial value of zero.
- The normal label for the X-register (x:) is replaced with the name of the variable or register being input and a question mark.

- Program execution halts, allowing you to key in or calculate a value.

When you press [R/S], the value in the X-register is automatically stored into the variable or register and program execution continues.

Pressing [EXIT] (if there is no menu displayed) cancels the INPUT function without storing any data. If you then press [R/S], the INPUT is resumed with the original value.

Example: Using INPUT. The formula for the surface area of a box is

$$Area = 2\,((length \times height) + (length \times width) + (height \times width)).$$

The following program uses INPUT to prompt for the values of *L*, *H*, and *W* and then calculates the surface area.

```
01 LBL "SAREA"
02 INPUT "L"
03 INPUT "H"
04 INPUT "W"
05 RCL× "L"
06 LASTX
07 RCL× "H"
08 RCL "H"
09 RCL× "L"
10 +
11 +
12 2
13 ×
14 END
```

Inputs each of the three variables.

Calculates *length* × *width*. The value of *W* is already in the X-register because it was the last value input.

Calculates *height* × *width*

Calculates *length* × *height*

Calculates sum of the products, multiplies by 2, and leaves the result in the X-register.

Key the program into your calculator.

[■] [GTO] [·] [·] [■] [PRGM]

```
00▶{ 0-Byte Prgm }
01 .END.
```

[■] [PGM.FCN] [■] [PGM.FCN] [LBL]
SAREA [ENTER]

```
01▶LBL "SAREA"
LBL RTN INPUT VIEW AVIEW XEQ
```

[INPUT] [ENTER] L [ENTER]

```
02▶INPUT "L"
LBL RTN INPUT VIEW AVIEW XEQ
```

[INPUT] [ENTER] H [ENTER]

```
03▶INPUT "H"
LBL RTN INPUT VIEW AVIEW XEQ
```

[INPUT] [ENTER] W [ENTER] [EXIT]

```
03 INPUT "H"
04▶INPUT "W"
```

[RCL] [×] [ENTER] L [ENTER]

```
04 INPUT "W"
05▶RCL× "L"
```

[■] [LASTx]

```
05 RCL× "L"
06▶LASTX
```

[RCL] [×] [ENTER] H [ENTER]

```
06 LASTX
07▶RCL× "H"
```

[RCL] [ENTER] H [ENTER]

```
07 RCL× "H"
08▶RCL "H"
```

[RCL] [×] [ENTER] L [ENTER]

```
08 RCL "H"
09▶RCL× "L"
```

[+]

```
09 RCL× "L"
10▶+
```

[+]

```
10 +
11▶+
```

2 [×]

```
12 2
13▶×
```

[EXIT]

Run the program to calculate the surface area of a box that is 4 × 3 × 1.5 meters.

[XEQ] SAREA

```
Y: 0.0000
L?0.0000
```

The program is prompting for a value of *L*. Key in the length (4) and press [R/S].

4 [R/S]

```
Y: 4.0000
H?0.0000
```

Key in the height (3) and press [R/S].

3 [R/S]

```
Y: 3.0000
W?0.0000
```

Key in the width (1.5) and press [R/S].

1.5 [R/S]

```
Y: 0.0000
X: 45.0000
```

The surface area is 45 square meters.

What is the surface area of a box that is twice as long? Run the program again. This time multiply the length by 2 and leave the other values as they are.

[XEQ] SAREA

```
Y: 45.0000
L?4.0000
```

2 [×] [R/S]

```
Y: 8.0000
H?3.0000
```

[R/S]

```
Y: 3.0000
W?1.5000
```

[R/S]

```
Y: 3.0000
X: 81.0000
```

The surface area is 81 square meters.

Using a Variable Menu

Using a *variable menu* may be the most efficient way for a program to input values for several variables. The VARMENU (*variable menu*) function creates a menu containing variable names. When the program stops, the menu is displayed allowing you to store, recall, and view variables.

The VARMENU function requires a global program label as a parameter. When a program executes VARMENU, the calculator searches for the specified program label. It then builds the variable menu using the MVAR (*menu variable*) instructions immediately following the specified label. (The calculator ignores MVAR instructions except when they are being read by the VARMENU function.*)

To store a value into a menu variable:

1. Key in or calculate the value.
2. Press the corresponding menu key.

To recall the value of a menu variable:

1. Press [RCL].
2. Press the corresponding menu key.

To view a menu variable without recalling it:

1. Press ■ (shift).
2. Press *and hold* the corresponding menu key. The message disappears when you release the key.

* The Solver and Integration applications also use variable menus defined with MVAR instructions.

To continue program execution:

- Press a menu key.
- *Or,* press [R/S].

If you continue by pressing a menu key, the name of the corresponding variable is stored into the Alpha register. Your program can use this information to determine which key was pressed. If you continue by pressing [R/S], the Alpha register is not altered.

To exit from a variable menu:

- Press [EXIT].
- *Or,* select an application menu ([SOLVER], [∫f(x)], [MATRIX], [STAT], or [BASE]).

Example: Using a Variable Menu. In the previous program, the INPUT function was used to prompt for three variables. By replacing lines 02, 03, and 04 with the following seven program lines, you can add a variable menu to the program.

```
02 MVAR "L"
03 MVAR "H"
04 MVAR "W"
```

Declares the menu variables following the global label.

```
05 VARMENU "SAREA"
06 STOP
07 EXITALL
```

Creates the variable menu and stops the program. When the program is restarted, the variable menu is exited.

```
08 RCL "W"
```

Since the variables in a variable menu can be entered in any order, there is no gaurantee that *W* will be in the X-register (as there was in the first program).

Edit the "SAREA" program. First delete lines 02, 03, and 04.

[PRGM] [GTO] [.] 4 [ENTER]

```
03 INPUT "H"
04▶INPUT "W"
```

[↓] [↓] [↓]

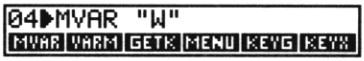
```
01▶LBL "SAREA"
02 RCL× "L"
```

Now insert the new program lines.

[PGM.FCN] [PGM.FCN] [▲] MVAR L

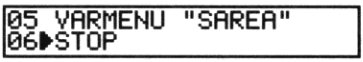
MVAR H

MVAR W



VARM SAREA

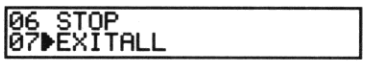
[EXIT] [R/S]



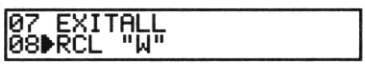
[CATALOG] FCN

Use the arrow keys to find the EXITALL function in the catalog.

[▼] ... [▼] EXITA



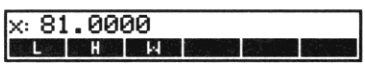
[RCL] W



[EXIT]

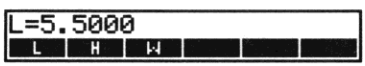
Now run the new version of the program.

[XEQ] SAREA

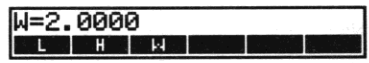


The variable menu is displayed, ready to use. Calculate the surface area of a box that is 5.5 × 2 × 3.75 cm.

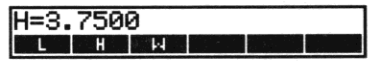
5.5 L



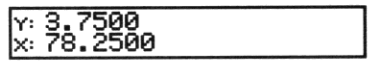
2 W



3.75 H



R/S



The surface area is 78.25 cm^2.

EXIT

Displaying Labeled Results (VIEW)

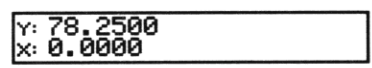
To display the contents of a variable or register use the VIEW function. VIEW creates a message that includes the variable or register name, an equal sign, and the data stored there. (Also refer to "Printing With VIEW and AVIEW" on page 132.)

For example, add these two lines to the end of the "SAREA" program.

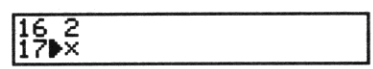
```
18 STO "SAREA"
19 VIEW "SAREA"
```

Line 18 stores the result into a variable named *SAREA*. Line 19 displays the contents of *SAREA*.

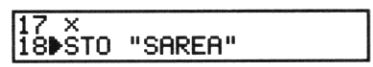
0 STO ENTER SAREA ENTER



PRGM GTO . 17 ENTER



STO SAREA



PGM.FCN VIEW SAREA

EXIT

```
Y: 78.2500
X: 0.0000
```

Now, run the program again using dimensions of 2 × 3 × 4 m.

XEQ SAREA

```
X: 0.0000
 L   H   W
```

2 L 3 W 4 H R/S

```
SAREA=52.0000
X: 52.0000
```

This time the answer is labeled for you. This technique is particularly useful when a program has several results.

Displaying Messages (AVIEW and PROMPT)

Messages are useful in programs to display descriptive prompts, output, and error conditions. For a program to display a message, it must:

1. Create the message in the Alpha register with an Alpha string.
2. Display the contents of the Alpha register.

To create a two line display, insert the *line feed* character (PUNC ▼ ↳) into the Alpha register as part of your message. When you execute AVIEW or PROMPT, characters following the line feed character are displayed on the second line of the display.

You can use more than one line feed character to produce multiline messages on the printer. However, since the calculator has a two-line display, anything following the second line feed character (within the same message) cannot be displayed.

The AVIEW Function. The AVIEW function displays the contents of the Alpha register. Depending on the status of flags 21 and 55, AVIEW may halt program execution or produce printer output. Refer to "Printing With VIEW and AVIEW" on page 132.

The PROMPT Function. The PROMPT function displays the contents of the Alpha register just as AVIEW does. However, PROMPT always halts program execution and only generates printer output in Normal and Trace printing modes.

Entering Alpha Strings Into Programs

An Alpha string entered as a program line—called an *Alpha constant*—is placed into the Alpha register when that line is executed. For a normal Alpha constant, like the one that follows, the Alpha string *replaces* the previous contents of the Alpha register.

```
01 "This is an"
```

If the *append symbol* precedes an Alpha string, the calculator appends the string to the current contents of the Alpha register.*

```
02 ⊢" Alpha String"
```

Append symbol

After executing these two program lines, the Alpha register contains:

```
This is an Alpha String
```

The "SMILE" program on page 139 uses program lines like this to create a special string in the Alpha register.

To key an Alpha string into a program:

1. Press ■ [ALPHA] to display the ALPHA menu.
2. Optional: press [ENTER] to insert the append symbol (⊢).
3. Type the string.
4. Press [ENTER] or ■ [ALPHA] to complete the string.

An Alpha string in a program may be up to 15 characters long. (The append symbol counts as a character.)

If the Alpha register fills up (44 characters), appending more characters pushes the left-most (oldest) characters out of the Alpha register.

* Note that some printers may not be able to print the append character.

This program displays three consecutive messages:

```
01 "Hello there,"
02 AVIEW
03 PSE
04 "this program"
05 AVIEW
06 PSE
07 "has 3 messages."
08 AVIEW
09 END
```

Without the PSE instructions (lines 03 and 06) the program would run too fast to see the first two messages. A PSE is not needed after the last AVIEW because the viewed information remains in the display after the program stops. Pressing a key during a PSE causes program execution to halt. Press [R/S] to resume program execution.

Printing During Program Execution

Printing is another important form of program output. For a complete description of print functions and modes, read chapter 7, "Printing."

Using Print Functions in Programs

When a print function (such as PRX, PRA, or PRV) is encountered in a running program, the calculator tests flags 21 and 55. In general, flag 21 (*printer enabled*) determines if printing is *desired* and flag 55 (*printer existence*) determines if printing is *possible*.

Flag 21	Flag 55	Result of Print Function
Clear	Set or clear	The print function is ignored and program execution continues with the next line.
Set	Clear	Program execution halts and displays `Printing Is Disabled`.
Set	Set	The print function is executed and the program continues.

Printing With VIEW and AVIEW

Like print functions, VIEW and AVIEW also test flags 21 and 55. In addition to performing their normal display functions, VIEW and AVIEW produce printed output if flags 21 and 55 are set.

To record results, set flag 21. If a program uses VIEW or AVIEW to display important results, set flag 21. Then if printing is enabled (flag 55 set), the information is printed.

If printing is disabled (flag 55 clear), the program stops so you can write down the displayed information. Press [R/S] to continue.

To display, but not record messages, clear flag 21. If flag 21 is clear, flag 55 is ignored by VIEW and AVIEW. The information is displayed and program execution continues.

Working With Alpha Data

This section describes the functions for manipulating data in the Alpha register. All of the techniques presented here can be executed manually; however, they are primarily meant for programming.

Moving Data Into and Out of the Alpha Register

In addition to keying data directly into the Alpha register or entering strings in programs, there are several ways to move data into and out of the Alpha register.

Storing Alpha Data. The ASTO (*Alpha store*) function copies the first six characters in the Alpha register into the specified variable or register. To execute the ASTO function:

1. If Alpha mode is not on, press ■ [ALPHA].
2. Press [ASTO]. (The [STO] key executes ASTO when Alpha mode is on.)

3. Specify where you want the string to be stored:
   - *In a storage register.* Key in the register number.
   - *In a variable.* Press a menu key to select the variable or use the ALPHA menu to type the name.
   - *In a stack register.* Press [.] followed by ST L, ST X, ST Y, ST Z, or ST T.

For example, to copy the first six characters of the Alpha register into the X-register, press [ALPHA] [ASTO] [.] ST X.

Recalling Data Into the Alpha Register. The ARCL (*Alpha recall*) function recalls data into the Alpha register, appending it to the current contents. To execute the ARCL function:

1. If Alpha mode is not on, press [ALPHA].
2. Press [ARCL]. (The [RCL] key executes ARCL when Alpha mode is on.)
3. Specify the storage register, variable, or stack register you want to recall. (Refer to step 3 above.)

If you recall a number into the Alpha register, it is converted to Alpha characters and formatted using the current display format. Recalling a matrix into the Alpha register recalls its descriptor (such as `[ 2×3 Matrix ]`).

When the Alpha register fills up, characters at the left end of the register (the "oldest" characters) are lost to make room for the new data.

To recall an integer into the Alpha register:

1. Place the number in the X-register.
2. Press [PGM.FCN] [▼] [▼] AIP (*Alpha append integer part*). The AIP function appends the integer part of the number in the X-register to the current contents of the Alpha register.

You can produce a similar result by using FIX 0 display format, clearing flag 29 (to remove the decimal point), and recalling a number using ARCL. A number recalled this way, however, may be rounded if the fractional part of the number is greater than or equal to 0.5.

To translate a number into a character:

1. Key in the character code (the allowed range is 0 through 255). Appendix E lists all of the display characters and their character codes.
2. Press [PGM.FCN] [▼] [▼] XTOA (*X to Alpha*).

If the X-register contains an Alpha string, the entire string is appended to the Alpha register.

If the X-register contains a matrix, the XTOA function uses each element in the matrix as a character code or Alpha string. XTOA begins with the first element (1:1) and continues rowwise (to the right) until it reaches the end of the matrix. If the Alpha register fills up, only the last 44 characters to be appended will remain.

The XTOA function is especially useful for building a graphics string in the Alpha register. Refer to the program on page 139.

To translate a character into its character code: Execute the ATOX (*Alpha to X*) function. ATOX converts the left-most character in the Alpha register into its character code (0 through 255) and returns the number to the X-register. The character is deleted from the Alpha register, shifting the rest of the string left one position. If the Alpha register is empty, ATOX returns zero.

For example, if the Alpha register contains `Janet`, executing ATOX deletes the `J` and returns its character code (74) to the X-register.

Searching the Alpha Register

To search the Alpha register for a character or string, use the POSA (*position in Alpha*) function. POSA searches the Alpha register for the *target* in the X-register. If a match is found, POSA returns the position number where the target was found (counting the left-most character as position 0). If a match is not found, POSA returns −1.

The target may be a character code or an Alpha string. POSA saves a copy of the target in the LAST X register.

Manipulating Alpha Strings

Once a string is in the Alpha register, there are several functions you can use to manipulate the data.

Finding the Length of an Alpha String. The ALENG (*Alpha length*) function returns to the X-register the number of characters in the Alpha register.

Shifting the Alpha Register. The ASHF (*Alpha shift*) function deletes the six left-most characters in the Alpha register. You may want to shift characters out of the Alpha register after using the ASTO function.

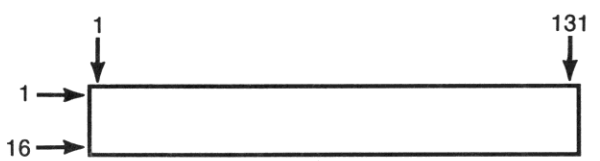
Rotating the Alpha Register. The AROT (*Alpha rotate*) function rotates the contents of the Alpha register by n characters (n is specified in the X-register). If n is positive, the rotation is to the left. If n is negative, the rotation is to the right.

Graphics

Using the functions PIXEL and AGRAPH (*Alpha graphics*), you can create graphics in the display of the HP-42S. The "DPLOT" and "PLOT" programs in the next chapter use the PIXEL function to produce graphs of functions (pages 156 and 160).

Turning On a Pixel in the Display

The PIXEL function turns on a pixel (one dot in the display) using the numbers in the X- and Y-registers. The x-value specifies the column (numbered from left to right; 1 through 131), and the y-value specifies the row (numbered from top to bottom; 1 through 16).



For a program to turn on a pixel in the display, it should:

1. Put the row number in the Y-register and the column number in the X-register.
2. Execute the PIXEL function ([PGM.FCN] [▼] [▼] [PIXEL]).

Executing PIXEL turns on the specified pixel and sets the message flags (flags 50 and 51). This allows subsequent PIXEL and AGRAPH instructions to add to the existing display.

To start with a clear display, execute CLLCD (*clear liquid crystal display*) before turning pixels on.

Drawing Lines in the Display

The PIXEL function can also be used to draw vertical and horizontal lines across the display. To draw a vertical line, use a negative x-value (-1 through -131). To draw a horizontal line, use a negative y-value (-1 through -16). If both numbers are negative, then PIXEL draws two lines—one vertical and one horizontal.

The plotting programs at the end of the next chapter use this feature of PIXEL to draw an x-axis.

Building a Graphics Image Using the Alpha Register

To create a graphics image in the display, a program should:

1. Create a string of characters in the Alpha register with each character specifying a column of eight pixels.
2. Specify where in the display the upper-left corner of the image should begin. Put that pixel-row number in the Y-register and the pixel-column number in the X-register.
3. Execute the AGRAPH function ([PGM.FCN] [▼] [▼] [AGRA]).

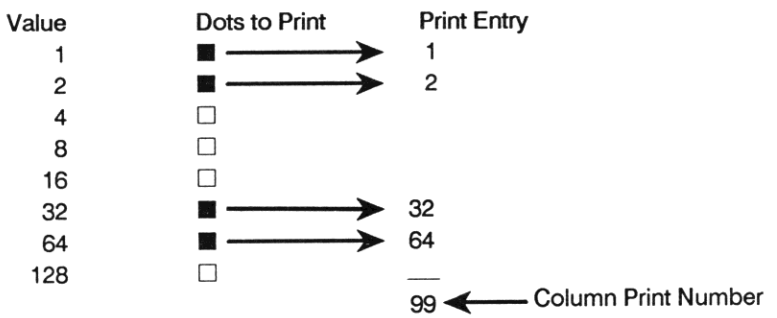
The status of flags 34 and 35 determine how the graphics image is displayed:

Flag 34	Flag 35	How the AGRAPH Image is Displayed
Clear*	Clear*	The image is merged with the existing display (logical OR).
Clear	Set	The image overwrites all pixels in that portion of the display.
Set	Clear	Duplicate "on" pixels get turned "off."
Set	Set	All pixels are reversed (logical XOR).

* Default setting.

Creating an Alpha String for AGRAPH. The AGRAPH function uses the character code of each character in the Alpha register as an eight-bit pattern for a column of pixels.

Each pixel in a column has a special value. Adding the values for all the pixels you want to display in a single column gives you the character code needed to produce that column.



To append the character to the Alpha register, key in the character code and then execute the XTOA function. You can type the character directly into the Alpha register if it is a typeable character. (Refer to the character table in appendix E.) For example, the character code 99 (calculated above) is the code for "c".

Example: Using Binary Mode to Calculate a Column Value. You can use the built-in Base application* to convert a column pattern into a character code. For example, select the Base application and Binary mode.

[■] [BASE] BINM

```
x: 110100
A...F HEXM DECM OCTM BIN■ LOGIC
```

Key in the column pattern above as a binary number. Start at the bottom, keying in a 0 for "off" pixels and a 1 for "on" pixels. (You can omit the leading zero if you want.)

01100011

```
x: 01100011_
A...F HEXM DECM OCTM BIN■ LOGIC
```

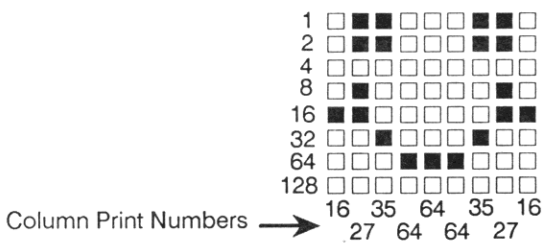
Display this number in Decimal mode.

DECM

```
x: 99.0000
A...F HEXM DEC■ OCTM BINM LOGIC
```

You don't have to use Decimal mode to use this number. While still in Binary mode, you could append the character to the Alpha register using the XTOA function.

Example: Displaying a Happy Face. The program below creates this happy face in the display.



Use the character table in appendix E to look up these character codes. If the table does not have keystrokes for a particular character (in this case character number 27), then append it to the Alpha register with the XTOA funciton. Refer to lines 03, 04, and 06 in the following program.

* Refer to chapter 16 for more information on the Base application.

```
01 LBL "SMILE"
02 "←"
03 27
04 XTOA
05 ⊢"#@@@#"
06 XTOA
07 ⊢"←"
08 5
09 62
10 CLLCD
11 AGRAPH
12 END
```

Character number 16.

Character number 27.

Character numbers 35, 64, 64, 64, and 35.

Character 27 (X-register still contains 27).

Character number 16.

Specifies the location of the image: row 5, column 62. (To key in the two numbers press 5 ENTER ◀ 62.)

Displays the image and stops.

Key in the "SMILE" program. (If you are still in the Base application from the previous example, press EXIT.)

■ GTO · · ■ PRGM

```
00▶{ 0-Byte Prgm }
01 .END.
```

■ PGM.FCN LBL SMILE ENTER

```
00 { 9-Byte Prgm }
01▶LBL "SMILE"
```

■ ALPHA ← ENTER*

```
01 LBL "SMILE"
02▶"←"
```

27

```
02 "←"
03▶27_
```

■ PGM.FCN ▼ ▼ XTOA

```
03 27
04▶XTOA
```

* After displaying the ALPHA menu (■ ALPHA), the keystrokes to type ← are: ▼ ←↑↓ ←.

■ ALPHA ENTER #@@@# *

```
05▶⊢"#@@@#_
( [ ≤ ←↑→↓ < = > MATH PUNC MISC
```

■ PGM.FCN ▼ ▼ XTOA

```
05 ⊢"#@@@#"
06▶XTOA
```

■ ALPHA ENTER ← ENTER

```
06 XTOA
07▶⊢"←"
```

5 ENTER ◆

```
08▶5
09 .END.
```

62

```
08 5
09▶62_
```

■ CLEAR ▼ CLLCD

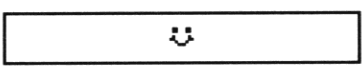
```
09 62
10▶CLLCD
```

■ PGM.FCN ▼ ▼ AGRA

```
10 CLLCD
11▶AGRAPH
```

Now exit from Program-entry mode and run the program.

EXIT XEQ SMILE



* After displaying the ALPHA menu and the append character (■ ALPHA ENTER), the keystrokes to type #@@@# are: ▼ MISC # MISC ▼ @ MISC ▼ @ MISC ▼ @ MISC #.

10

Programming Techniques

This chapter covers functions and techniques for writing more sophisticated programs. You'll learn how to use:

- GTO (*go to*) and XEQ (*execute*) instructions to cause program branching to execute subroutines and other programs.
- The programmable menu to create *menu-driven* programs.
- Conditional tests and counters to create program *loops* (routines that repeat themselves).
- Tests and comparisons to make decisions and cause program branching.

Branching

Branching occurs whenever the program pointer moves to a line other than the "next" line—that is, whenever program instructions are not executed sequentially. The two primary functions for branching are GTO and XEQ.

Often flag tests and comparisons are followed by branching instructions that are executed according to the result of the test or comparison.

Branching to a Label (GTO)

Labels can be considered *destinations* for branching instructions. As explained in chapter 8, global labels can be accessed from anywhere in memory and local labels can be accessed only from within their own program.

There are three programmable forms of GTO instructions:

- GTO *nn* for branching to a local numeric label (where *nn* is the label number).
- GTO *label* for branching to a local Alpha label (where *label* is a single letter A through J or a through e).
- GTO "*label*" for branching to a global label (where *label* is the Alpha label).

Here are a few examples:

Example Instruction:	**Description (Keys):**
GTO 03	Branches to LBL 03 (GTO 03).
GTO A	Branches to LBL A (GTO ENTER A ENTER).
GTO "AREA"	Branches to LBL "AREA" (GTO AREA).

Executing GTO in a Program. In a running program, a GTO instruction causes program execution to branch to the specified label and continue running at that line.

Executing GTO From the Keyboard. Executing a GTO instruction from the keyboard moves the program pointer to the corresponding label. No program lines are executed.

Indirect Addressing With GTO. The following examples show how indirect addressing can be used with GTO instructions. That is, the label to be branched to is specified in a variable or register.

Example Instruction:	Description (Keys):
`GTO IND 12`	Branches to the label specified in storage register R_{12} ([GTO] [.] [IND] 12). For example, if R_{12} contains the string "AREA", then program execution branches to `LBL "AREA"`.
`GTO IND "ABC"`	Branches to the label specified in the variable *ABC* ([GTO] [.] [IND] [ABC]). For example, if *ABC* contains the number 17, then program execution branches to `LBL 17`.
`GTO IND ST X`	Branches to the label specified in the X-register ([GTO] [.] [IND] [.] [ST X]). For example, if the X-register contains the number 96, then program execution branches to `LBL 96`.

Calling Subroutines (XEQ and RTN)

The GTO function, described above, is used to make a simple program branch. XEQ is used in much the same way with one important difference: *after* an XEQ instruction has transferred execution to the specified label, the next RTN (*return*) or END instruction causes the program to branch *back* to the instruction that immediately follows the XEQ instruction.

XEQ instructions are *subroutine calls*. A subroutine call is not complete until a RTN or END has been executed to return program execution to the line following the XEQ instruction.

XEQ is also used to run programs from the keyboard ([XEQ]).

Example: GTO versus XEQ. Consider the following two programs. If you execute the first program ([XEQ] [PRGM]), `TONE 0` never executes because the GTO instruction branches to the second program. Program execution halts when the END is reached in the second program.

```
01 LBL "PRG1"
02 GTO "PRG2"
03 TONE 0
04 END
```

```
01 LBL "PRG2"
02 TONE 9
03 END
```

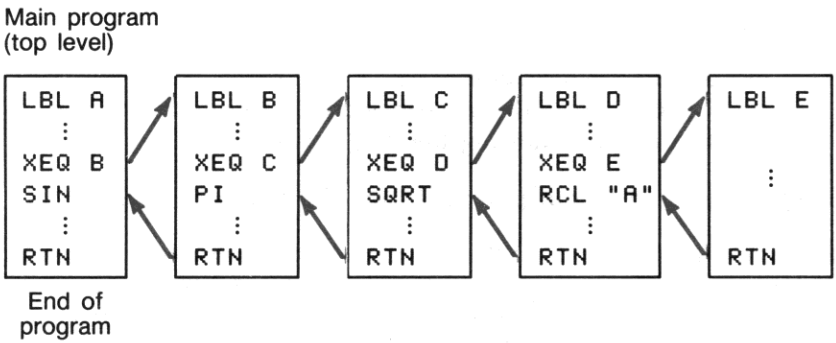
However, if you replace line 02 of the first program with an XEQ instruction (`XEQ "PRG2"`), both TONEs will sound. When the END is encountered in the second program, execution returns to the line immediately following the XEQ. Program execution halts at the END in the first program.

```
01 LBL "PRG1"
02 XEQ "PRG2"
03 TONE 0
04 END
```

```
01 LBL "PRG2"
02 TONE 9
03 END
```

Subroutine Return Locations. When an XEQ instruction calls a subroutine, the HP-42S remembers the location of that XEQ instruction so that execution can return there when the subroutine is completed.

For example, this illustration shows how the calculator *nests* subroutines by remembering return locations. The HP-42S can remember up to eight pending return locations.



Loss of Subroutine Returns. Pending return locations are lost under the following conditions:

- If there are already eight pending return locations when another subroutine or program is called with an XEQ, the first (oldest) return location is lost.* In this case, program execution never returns to the first XEQ that called a subroutine. Instead, execution halts when the first subroutine is finally completed because there are no further return locations.
- All pending return locations are lost when you execute any program from the keyboard or perform any other operation (while program execution is halted) that alters the program pointer. Pressing ■ SST or R/S does not cause return locations to be lost.

The Programmable Menu

The HP-42S has a programmable menu which is used to cause program branching. The MENU function selects the programmable menu. The menu is displayed when the program stops. You can define each key in the menu so that when the key is pressed, a particular GTO or XEQ instruction executes. You can even define ▲, ▼, and EXIT.

To define a menu key:

1. Enter a string into the Alpha register. This is the text that appears in the menu label above the key. (The Alpha register is not used when defining ▲, ▼, or EXIT.)
2. Execute KEYG (*on key, go to*) or KEYX (*on key, execute*). (These functions are in the last row of the PGM.FCN menu; press ■ PGM.FCN ▲.)
3. Specify which key you want to define:
   - Press Σ+, 1/x, √x, LOG, LN, XEQ, ▲, ▼, or EXIT.
   - *Or,* key in the key number, 1 through 9.

* The Solver and Integration applications also create return locations. If the calculator loses one of these returns, program execution stops and an error message is displayed.

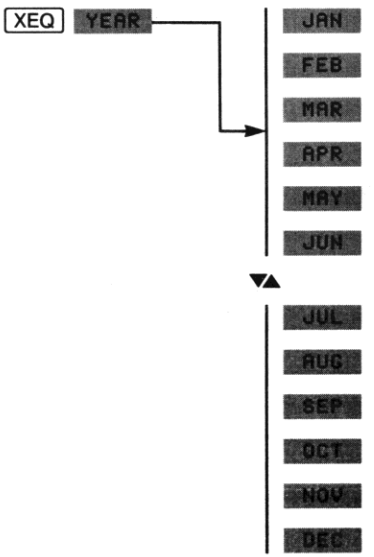
4. Specify a program label using *one* of these methods:
   - Select an existing global label by pressing the corresponding menu key.
   - Use the ALPHA menu to type an Alpha label (local or global): [ENTER] *label* [ENTER].
   - Key in a two-digit numeric label.

Repeat this procedure for each menu key you want to define. Defining a key replaces any previous definition that may exist for that key.

To display the programmable menu: Execute the MENU function (press [PGM.FCN] [▲] MENU).

To clear all programmable menu key definitions: Execute the CLMENU (*clear menu*) function (press [CLEAR] [▼] CLMN).

Example. The program segment listed below shows how the programmable menu can be used to emulate this menu:



```
01 LBL "YEAR"
02 LBL A
03 "JAN"
04 KEY 1 XEQ 01
05 "FEB"
06 KEY 2 XEQ 02
07 "MAR"
08 KEY 3 XEQ 03
09 "APR"
10 KEY 4 XEQ 04
11 "MAY"
12 KEY 5 XEQ 05
13 "JUN"
14 KEY 6 XEQ 06
```

Defines the first row of the "YEAR" menu. A different subroutine is executed for each month. The routines for the first six months are labeled with local labels 01 through 06.

```
15 KEY 7 GTO B
16 KEY 8 GTO B
17 KEY 9 GTO 99
```

Defines the [▲], [▼], and [EXIT] keys. The [▲] and [▼] keys are defined to go to the same program label (LBL B) because this is a two-row menu; either key should display the second row. The [EXIT] key is defined to cause a branch to a routine that exits the menu.

```
18 MENU
19 LBL 20
20 STOP
21 GTO 20
```

The programmable menu is selected and the program stops. Because of this little loop, pressing [R/S] keeps the program at line 20.

```
22 LBL B
23 "JUL"
24 KEY 1 XEQ 07
25 "AUG"
26 KEY 2 XEQ 08
27 "SEP"
28 KEY 3 XEQ 09
29 "OCT"
30 KEY 4 XEQ 10
31 "NOV"
32 KEY 5 XEQ 11
33 "DEC"
34 KEY 6 XEQ 12
```

Defines the menu keys for the second row of the "YEAR" menu.

```
35 KEY 7 GTO A
36 KEY 8 GTO A
```

Defines the [▲] and [▼] to return to the first row of the menu. The [EXIT] key does not need to be define again. The definition made at line 17 is still in effect.

```
37 LBL 21
38 STOP
39 GTO 21
```

Stops the program. The programmable menu is still selected (line 18).

```
40 LBL 99
41 CLMENU
42 EXITALL
43 RTN
```

The menu definitions are cleared and the menu is exited. If this program was called as a subroutine from another program, execution returns to that program.

```
44 LBL 01
      ⋮
```

The rest of the program consists of the subroutines for each month (LBL 01 ... RTN, LBL 02 ... RTN, and so on). For example, you might want to create a message in each of these subroutines that displays the full name of the month and the number of days in that month.

Many examples in the *HP-42S Programming Examples and Techniques* manual (part number 00042-90020) use the programmable menu.

Local Label Searches

Searches for local labels occur only within the current program. To find a local label, the calculator first searches sequentially downward through the current program, starting at the location of the program pointer. If the specified label is not found before reaching the end of the program, the calculator continues the search from the beginning of the program.

A local label search can consume a significant amount of time, depending on the length of the current program and the distance to the label. To minimize searching time, the calculator remembers the distance from the GTO or XEQ instruction to the specified local label.* This eliminates the searching time for subsequent executions of that same GTO or XEQ instruction.

Global Label Searches

When the calculator searches for a global label, the search begins with the *last* global label (bottom of program memory) and proceeds *upward,* stopping at the first label that matches the specified label. The search is in the same order as the labels are listed in the program catalog.

Conditional Functions

Flag tests and comparisons are *conditional functions.* They express a proposition that is either true or false depending on current conditions.

- Executing a conditional function from the keyboard generates a message: `Yes` if the proposition is currently true, or `No` if the proposition is currently false.
- Executing a conditional function in a program causes a program branch using the *do-if-true* rule. That is, the program line immediately following the conditional is executed *only* if the condition is true. If the condition is false, the next line is *skipped.* That is, DO the next instruction IF the condition is TRUE.

* The distance to the label is stored internally as part of the GTO or XEQ instruction. If this distance is greater than 4,096 bytes in either direction (128 bytes for short form lables; LBL 00 through LBL 14), the calculator cannot store the distance and a search must take place for each execution of the instruction.

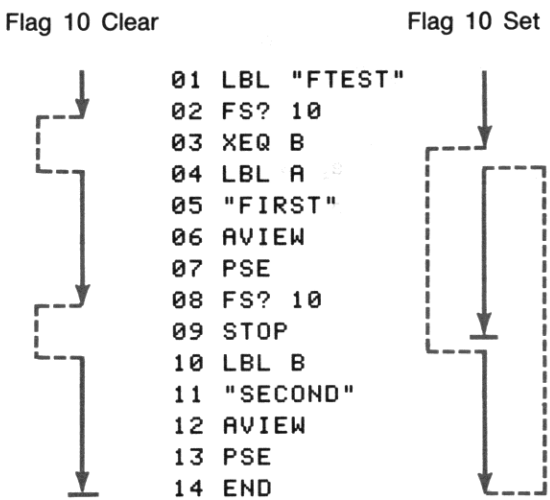
Flag Tests

The following table shows the four flag test functions and how each causes program branching (a skipped line) based on the status of the flag being tested. (These functions are in the FLAGS menu.)

Flag Test	If Flag Is Set	If Flag Is Clear
FS?	Execute the next program line.	Skip the next program line.
FC?	Skip the next program line.	Execute the next program line.
FS?C*	Clear flag and execute the next program line.	Clear flag and skip the next program line.
FC?C*	Clear flag and skip the next program line.	Clear flag and execute the next program line.

* This function can only be used with flags 00 through 35 and 81 through 99.

The following program demonstrates a subroutine call (line 03) and flag tests (lines 02 and 08). If flag 10 is clear, `FIRST` is displayed, and then `SECOND`. If flag 10 is set, the order of the messages is reversed.



Comparisons

To compare the X-register with zero:

1. Press ■ PGM.FCN ▼ X?0 .
2. Press X=0? , X≠0? , X<0? , X>0? , X≤0? , or X≥0? .

To compare the X-register with the Y-register:

1. Press ■ PGM.FCN ▼ X?Y .
2. Press X=Y? , X≠Y? , X<Y? , X>Y? , X≤Y? , or X≥Y? .

If you execute one of these functions from the keyboard, the calculator displays Yes or No, indicating the result of the test. If a program executes one of these functions, the calculator follows the do-if-true rule.

Testing the Data Type

The following four functions test the type of data in the X-register. They also follow the do-if-true rule for program execution.

Function	Test Proposition
REAL?	Does the X-register contain a real number?
CPX?	Does the X-register contain a complex number?
MAT?	Does the X-register contain a matrix?
STR?	Does the X-register contain an Alpha string?

Bit Test

The BIT? (*bit test*) function tests a single bit of a number. If the x^{th} bit of y is a 1, then the test is true. Refer to chapter 16 for more information on the Base application and logic functions.

Looping

A loop is a sequence of program instructions that starts with a label and ends with a branch back to that label. An infinite loop is the simplest kind. Once started, this program runs until you stop it with [R/S] or [EXIT].

```
01 LBL "LOOP"
02 BEEP
03 GTO "LOOP"
04 END
```

Looping Using Conditional Functions

When you want to perfom an operation until a certain condition is met, but you don't know how many times the loop must repeat, you can create a loop with a conditional test and a GTO instruction.

For example, the following program loops until the RAN (*random number*) function returns a number that is at least 0.9. That is, the loop repeats if the random number is less than 0.9.

```
01 LBL "RANDOM"
02 LBL 01
03 0.9
04 RAN
05 X<Y?
06 GTO 01
07 END
```

Why does this program have two labels? Since the HP-42S only has to search for a local label once, the loop executes faster by branching to a local label. (Refer to "Local Label Searches" on page 148.) What's more, using a local label and corresponding GTO instruction (rather than branching to the global label) saves five bytes of program memory.

Loop-Control Functions

When you want to execute a loop a specific number of times, you can use special functions for that purpose—ISG (*increment, skip if greater*) and DSE (*decrement, skip if less than or equal*). Both functions (located in the PGM.FCN menu) take a parameter identifying the variable or register containing the number that controls the looping.

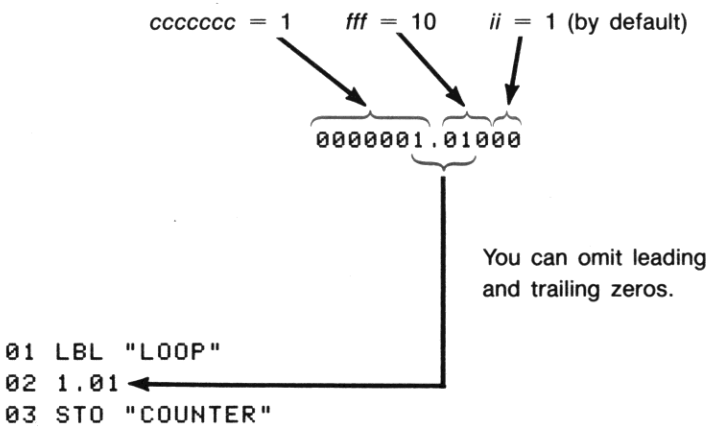
The format of the loop-control number is *ccccccc.fffii*, where:

- *ccccccc* is the current counter value. Executing ISG or DSE increments or decrements *ccccccc* by the value of *ii*.
- *fff* is the final counter value.
- *ii* is the increment/decrement value. If *ii* is 00 (or unspecified), the calculator uses a default value of 01.

Executing ISG increments *ccccccc* by *ii*, and then compares the resulting value of *ccccccc* with *fff*. If *ccccccc* is greater than *fff*, the next program instruction is skipped.

Executing DSE decrements *ccccccc* by *ii*, and then compares the resulting value of *ccccccc* with *fff*. If *ccccccc* is less than or equal to *fff*, the next program instruction is skipped.

Example: Using the ISG Function. The following program uses ISG to loop 10 times. The loop counter is stored into a variable named *COUNTER* and is interpreted by the ISG function like this:



```
04 LBL 01
05 VIEW "COUNTER"
06 PSE
07 ISG "COUNTER"
08 GTO 01
09 "DONE"
10 AVIEW
11 END
```

Controlling the CUSTOM Menu

If flag 27 is set when a program stops, the CUSTOM menu is displayed. Before displaying the menu, however, the calculator also checks flag 72.* If flag 72 is clear (indicated by KEY■ in the MODES menu), the CUSTOM menu displays menu assignments you have made. If flag 72 is set (indicated by LCLB■ in the MODES menu), the CUSTOM menu displays keys for executing local labels (page 301).

Example Programs

The programs in this section use many of the functions and techniques presented in chapters 8, 9, and 10. By examining them and using them, you should gain an even better understanding of programming. For many more programming examples, refer to the *HP-42S Programming Examples and Techniques* manual (part number 00042-90020).

The Display Plot Program ("DPLOT")

The "DPLOT" program plots a function in the display of the calculator. The function that you plot is entered into the calculator as a program. There are two general forms for a function program:

* The calculator also checks flag 72 when you use ■ CUSTOM to display the CUSTOM menu.

- *As f(x)*, where the program returns a value using an input value in the X-register. For example, to plot a sine curve ($f(x) = \sin x$), use a program like this:

```
01 LBL "SINE"
02 SIN
03 END
```

- *As a Solver program.* If the program uses menu variables, it is assumed to be written in the proper form for use with the Solver. Refer to "Writing a Program for the Solver" on page 179.

The name of the function is stored in a variable named *FCN*. Since Alpha strings stored in variables are limited to six characters, the global label that you use to identify the function cannot be longer than six characters.

You can determine what portion of the function is plotted by entering the limits of the plot:

YMIN = bottom of the display
YMAX = top of the display
XMIN = left end of the display
XMAX = right end of the display

You can also specify where you would like the *x*-axis to appear. Usually, the axis is at $y = 0$. If you don't want an axis, specify a *y*-value that is less than *YMIN* or greater than *YMAX*.

To use the "DPLOT" program:

1. Key the "DPLOT" program into your calculator. (The "DPLOT" program uses 234 bytes of program memory.)
2. Key in a program for the function you want to plot.
3. Press [XEQ] DPLOT. The program displays a variable menu containing *YMIN*, *YMAX*, *AXIS*, *XMIN*, and *XMAX*. Store a value into each variable: key in a number and then press the corresponding menu key.
4. Press [R/S]. The program displays the current function name stored in *FCN* (if there is one) along with the Alpha menu.
5. If necessary, type the name of the function you want to plot.

6. Press [R/S]. If the function does not use menu variables, plotting begins.
7. If the function does use menu variables, the program stops and displays the variable menu. Using the variable menu:
   a. Store a value into each of the known variables: key in a number and then press the corresponding menu key.
   b. Press a menu key to select the plot variable. Plotting begins.

When the plot is finished, the program prints a copy of the display (if printing is enabled).

The example on page 185 uses "DPLOT" to plot a function for the Solver.

Program:	**Comments:**

`01 LBL "DPLOT"`
`02 MVAR "YMIN"`
`03 MVAR "YMAX"`
`04 MVAR "AXIS"`
`05 MVAR "XMIN"`
`06 MVAR "XMAX"`
	Declares the menu variables.

`07 LBL A`
`08 VARMENU "DPLOT"`
`09 "Ready"`
`10 PROMPT`
	Selects the variable menu, displays a `Ready` message, and stops the program.

`11 CLA`
`12 SF 25`
`13 RCL "FCN"`
`14 CF 25`
`15 STR?`
`16 ARCL ST X`
	Recalls the current function name (if there is one) into the Alpha register.

`17 AON` `18 STOP`	Turns on the ALPHA menu and stops the program so a function name can be entered or changed.

```
19 AOFF
20 ALENG
21 X=0?
22 GTO A
23 ASTO "FCN"
```

Turns off the ALPHA menu and tests the length of the Alpha register. If the Alpha register is empty, execution returns to the first variable menu. Otherwise, the function name is stored into *FCN*.

```
24 CLA
25 CF 81
26 SF 25
27 VARMENU IND "FCN"
28 FC?C 25
29 SF 81
```

Selects the variable menu for the function. If there are no menu variables, flag 81 is set.

```
30 FC? 81
31 STOP
32 EXITALL
33 ALENG
34 X=0?
35 SF 81
36 ASTO 03
```

Stops to display the variable menu (if flag 81 is clear). Tests the Alpha register to see if a plot variable has been selected. If not, flag 81 is set. The variable name is stored into R_{03}.

```
37 15
38 RCL "YMAX"
39 RCL- "YMIN"
40 ÷
41 STO 00
```

Calculates the *y*-value of one pixel.

```
42 RCL "XMIN"
43 STO 01
44 1.131
45 STO 02
```

Stores the first *x*-value and a loop counter. (There are 131 pixels across the display.)

```
46 CLLCD
47 XEQ "AXIS"
```

Clears the display and draws an axis.

```
48 LBL 01
49 RCL 01
50 FC? 81
51 STO IND 03
52 XEQ IND "FCN"
```

Recalls the current *x*-value. If flag 81 is clear, the *x*-value is stored into the plot variable. The function is then evaluated using the current *x*-value.

```
53 XEQ 02
54 RCL 02
55 PIXEL
```

The value of the function is converted into a pixel number.

```
56 RCL "XMAX"
57 RCL- "XMIN"
58 131
59 ÷
60 STO+ 01
```

The x-value is incremented.

```
61 ISG 02
62 GTO 01
63 PRLCD
64 RTN
65 GTO A
```

If the plot is done, the display is printed and the program stops. Line 65 allows the program to be restarted by pressing [R/S].

```
66 LBL 02
67 RCL- "YMIN"
68 RCL× 00
69 16
70 -
71 X>0?
72 CLX
73 ABS
74 RTN
```

Calculates a pixel number for the given y-value.

```
75 LBL "AXIS"
76 RCL "AXIS"
77 XEQ 02
78 +/-
79 1
80 PIXEL
81 END
```

Draws an x-axis.

The Printer Plot Program ("PLOT")

The "PLOT" program plots a function on the HP 82240A printer. The plot is created in sections. Each section is plotted in the display and then printed. The result is a continuous plot of the function on a strip of paper. (The x-axis runs lengthwise on the paper.)

Before plotting a function, you must write a program that expresses the function. The name of the function is stored into a variable named *FCN*. Since Alpha strings stored in variables are limited to six characters, the global label that you use to identify the function must be six or fewer characters.

You can determine what portion of the function is plotted by entering the limits of the plot:

YMIN = left edge of paper
YMAX = right edge of paper
XMIN = beginning x-value
XMAX = ending x-value
XINC = increment of x-values

The x-values are printed at increments determined by *XINC*. If you do not want these labels on your plot, set flag 00.

You can specify where you would like the x-axis to appear. Usually, the axis is at $y = 0$. If you do not want an axis, set flag 01.

To use the "PLOT" program:

1. Key the "PLOT" program into your calculator. (The "PLOT" program uses 337 bytes of program memory.)
2. Key in a program for the function you want to plot.
3. Press [XEQ] [PLOT]. The program displays a variable menu containing *YMIN*, *YMAX*, *AXIS*, *XMIN*, *XMAX*, and *XINC*. Store a value into each variable: key in a number and then press the corresponding menu key.
4. Press [R/S]. The program displays the current function name stored in *FCN* (if there is one) along with the Alpha menu.
5. If necessary, type the name of the function you want to plot.
6. Press [R/S] to begin the plot.

```
01 LBL "PLOT"
02 MVAR "YMIN"
03 MVAR "YMAX"
04 MVAR "AXIS"
05 MVAR "XMIN"
06 MVAR "XMAX"
07 MVAR "XINC"
```

Declares the menu variables.

```
08 LBL A
09 VARMENU "PLOT"
10 STOP
```

Selects the variable menu and stops the program.

```
11 EXITALL
12 XEQ 07
```

Exits from the variable menu and inputs a function name.

```
13 PRON
14 ADV
15 "Plot of:"
16 PRA
17 ADV
18 SF 12
19 CLA
20 ARCL "FCN"
21 PRA
22 ADV
23 CF 12
24 PRV "YMIN"
25 PRV "YMAX"
26 PRV "AXIS"
27 PRV "XMIN"
28 PRV "XMAX"
29 PRV "XINC"
30 ADV
31 "← YMIN"
32 ⊢"        YMAX →↳"
33 PRA
```

Prints the header information.

```
34 130
35 RCL "YMAX"
36 RCL- "YMIN"
37 ÷
38 STO 00
```

Calculates the y-value of one pixel.

```
39 RCL "XMIN"
40 STO 01
```

Stores the first x-value.

```
41 LBL 00
42 CLLCD
```

Clears the display.

```
43 FC? 00
44 XEQ 05
```

Labels the x-increment if flag 00 is clear.

```
45 FC? 01
46 XEQ 06
```

Draws an axis if flag 01 is clear.

```
47 1.016
48 STO 02
```

Stores a loop counter into R_{02}. (There are 16 rows of pixels in the display.)

```
49 LBL 01
50 RCL "FCN"
51 STR?
52 XEQ 04
```

Plots the current point.

```
53 RCL "XINC"
54 16
55 ÷
56 STO+ 01
```

Increments the x-value.

```
57 RCL "XMAX"
58 RCL 01
59 X>Y?
60 GTO 03
```

Goes to LBL 03 if the plot is done.

```
61 ISG 02
62 GTO 01
63 PRLCD
64 GTO 00
```

Prints the display if all 16 values have been plotted.

```
65 LBL 03
66 PRLCD
67 RTN
68 GTO A
```

Prints the final display. Line 68 allows the program to be restarted by pressing [R/S].

```
69 LBL 04
70 RCL 01
71 XEQ IND ST Y
72 SF 24
73 RCL- "YMIN"
74 RCL× 00
75 1
76 +
77 CF 24
78 RCL 02
79 X<>Y
80 X>0?
81 PIXEL
82 RTN
```

Evaluates the function at the current x-value and then plots the appropriate pixel.

```
83 LBL 05
84 CF 21
85 CLA
86 ARCL 01
87 AVIEW
88 SF 21
89 RTN
```

Puts an x-value into the display to label the x-axis.

```
90 LBL 06
91 1
92 RCL "AXIS"
93 RCL- "YMIN"
94 RCL× 00
95 +/-
96 1
97 -
98 PIXEL
99 +/-
100 2
101 -
102 "xxxxx"
103 AGRAPH
104 RTN
```

Plots an x-axis. Note that line 102 is a string of multiply characters ([■] [ALPHA] [×] [×] [×] [×] [×] [ENTER]).

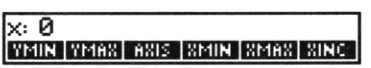
```
105 LBL 07
106 CLA
107 SF 25
108 RCL "FCN"
109 CF 25
110 STR?
111 ARCL ST X
112 AON
113 STOP
114 AOFF
115 ASTO "FCN"
116 END
```

Recalls the current function name (if there is one) into the Alpha register. Turns on the ALPHA menu and stops the program. When the program continues (when R/S is pressed), the function name is stored into *FCN*.

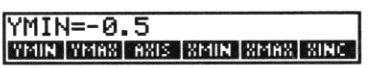
Example: Using the Printer Plot Program. Key in the "PLOT" program listed above and the program "MISCFN" below. Plot the function with *YMIN* = −0.5, *YMAX* = 2, *AXIS* = 0, *XMIN* = −360, *XMAX* = 360, and *XINC* = 45.

```
01 LBL "MISCFN"
02 ENTER
03 ENTER
04 360
05 ÷
06 X<>Y
07 3
08 ×
09 SIN
10 ×
11 1
12 +
13 END
```

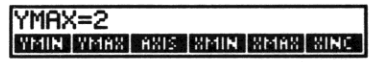
DISP ALL XEQ PLOT



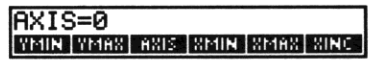
.5 +/- YMIN



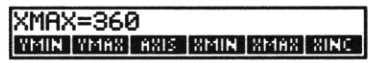
2 YMAX



0 AXIS

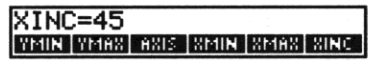


360 XMAX

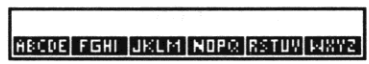


+/- XMIN

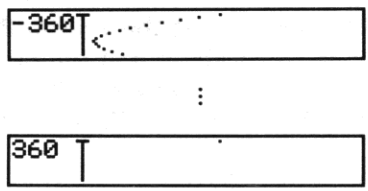
45 XINC



R/S

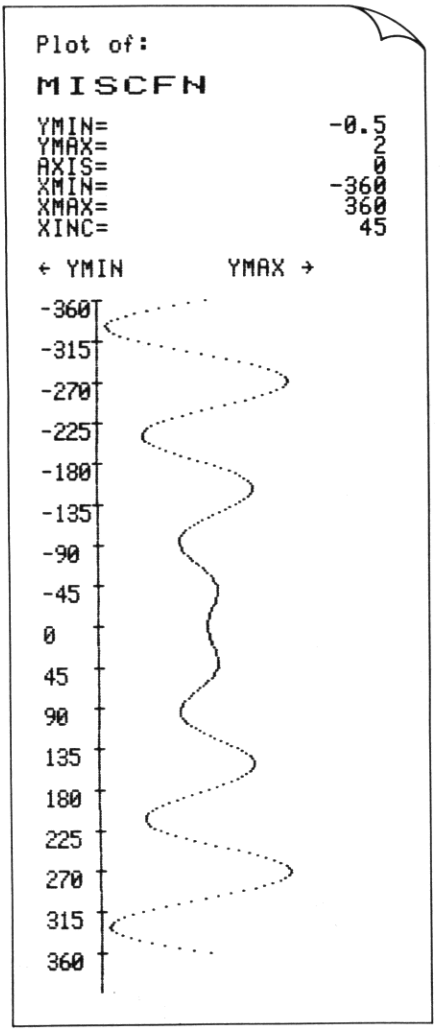


MISCFN R/S



The printer output is shown on the following page.

Printer Output:



11

Using HP-41 Programs

All programmable functions of the HP-41C and HP-41CV calculators have been built into the HP-42S. This means that programs written for these HP-41* calculators will run on the HP-42S.

In addition to the HP-41C/CV function set, several new functions have been added to further enhance the programming capabilities of the HP-42S. As you become more familiar with programming, you will probably want to modify your favorite HP-41 programs to take advantage of the expanded function set of the HP-42S.

In this chapter you'll learn about:

- Special considerations you may need to make when running some HP-41 programs.
- Reading HP-41 program listings and keying programs into the HP-42S.
- Enhancing HP-41 programs.

Important Differences

While the HP-42S fully supports the function set of the HP-41C/CV calculators, there are some important differences that you should not overlook. Under most circumstances, these differences will add to the accuracy or capability of an existing HP-41 program. Some HP-42S operations, however, may need to be disabled so operation more closely emulates the HP-41.

* "HP-41" is used in this chapter to refer to the HP-41C and HP-41CV calculators. Not all extended functions built into the HP-41CX calculator are supported by the HP-42S.

HP-41 User Keyboard

The CUSTOM menu in the HP-42S provides capabilities that are similar to the User keyboard on the HP-41. That is, you can:

- Assign functions and programs to the CUSTOM menu.
- Use the CUSTOM menu to execute local labels in the current program.

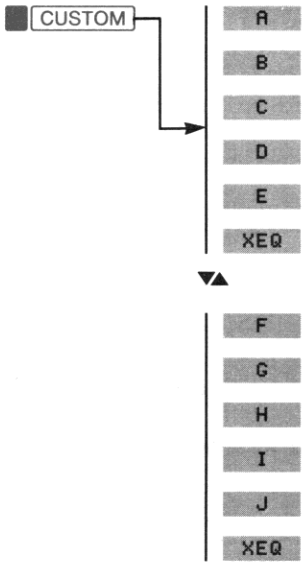
Flag 27, which is used on the HP-41 to control the User keyboard, is used to control the CUSTOM menu. In general, setting flag 27 is equivalent to pressing ■ [CUSTOM]. Clearing flag 27 is equivalent to pressing [EXIT] when the CUSTOM menu is displayed.

To use CUSTOM menu assignments:

1. If necessary, press ■ [MODES] [▼] KEY (*key assignments*) to select Key-assignment mode. The calculator selects this mode automatically each time you make an assignment to the CUSTOM menu (■ [ASSIGN]). The KEYASN function clears flag 72.
2. Press ■ [CUSTOM] or ■ [FLAGS] SF 27 to display the CUSTOM menu.

To use CUSTOM menu for executing local labels:

1. If necessary, press ■ [MODES] [▼] LCLBL (*local label*) to select *Local-label* mode. The LCLBL function sets flag 72.
2. Press ■ [CUSTOM] or ■ [FLAGS] SF 27 to display the CUSTOM menu.



Pressing A through J executes the instructions XEQ A through XEQ J. Use the shift key (■) to execute XEQ a through XEQ e (■ A through ■ E).

If you are using an HP-41 program that uses local Alpha labels, the instructions may say something like "Press [B]." When you're running the program, remember this means to press B. Similarly, if the instructions say "Press [b]," then press ■ B.

Statistical Operations

Statistical operations on the HP-42S have been expanded (beyond the capabilities of the HP-41) to include curve fitting and forecasting. These enhanced features require the use of seven more summation coefficients than the HP-41 uses.

To use only 6 summation coefficients (like the HP-41): Press ■ [STAT] [▼] LINΣ.

To use all 13 summation coefficients (default): Press ■ [STAT] [▼] ALLΣ.

Printer Interface

Because the HP-42S uses a one-way infrared printer interface, it cannot tell if a printer is receiving the infrared signal. It's up to you to tell the calculator if a printer is available.

To enable printing: Press ■ PRINT ▲ PON .

To disable printing: Press ■ PRINT ▲ POFF .

Refer to chapter 7, "Printing," for more information.

The Alpha Register

The Alpha register in the HP-42S is 44 characters long, which is 20 characters longer than the Alpha register in the HP-41. Programs that specifically require the Alpha register to be 24 characters long may not produce the desired output.

Range of Numbers

The HP-42S uses 15 digits (a 12-digit mantissa and a 3-digit exponent of ten) to represent all real numbers. The HP-41, however, uses a 10-digit mantissa and a 2-digit exponent. Therefore, because of this increased range, calculations that generate an "OUT OF RANGE" error on the HP-41 may not be out of range on the HP-42S.

Note that the HP-42S returns an `Out of Range` error for the tangent of 90°. The HP-41 returns $9.999999999 \times 10^{99}$.

Data Errors and the Real-Result Flag

Because of its complex-number capabilities, the HP-42S can return results for calculations that would not work on the HP-41. The HP-42S automatically returns a complex number for calculations such as:

- Square root of a negative number.
- Logarithm of a negative number.
- Arc sine or arc cosine of a number whose absolute value is greater than 1.

To disable complex results for real-number operations: Press [■] [MODES] [▼] RRES (*real results only*). This function sets flag 74, which prevents the calculator from producing a complex result. Attempting an operation that would normally return a complex number, displays `Invalid Data`.

Note that flag 74 is only observed if the inputs for a function are real numbers. That is, if one or more inputs for a function are already complex, the result will be complex, regardless of the state of flag 74.

To enable complex results for real-number operations: Press [■] [MODES] [▼] CRES (*complex-result enable*). This function clears flag 74 (default).

The Display

The HP-42S uses a two-line, 22-character display while the HP-41 uses a single-line, 12-character display. Therefore, programs that specifically format output for the HP-41 display may not produce the desired displays on the HP-42S.

The HP-42S does not *scroll* the display as the HP-41 does. The calculator indicates when a number is too large for the display by showing the ... (ellipsis) character. Press and hold [■] [SHOW] to see the full-precision value for the number in the X-register.

Keystrokes

For the most part, the keystroke sequences on the HP-42S are similar to the HP-41. The following exceptions are worth noting:

- Alpha characters are typed with the ALPHA menu (page 37).
- Indirect addressing on the HP-41 uses the shift ([■]) key. The HP-42S, on the other hand, uses [·] or [·] IND to specify indirect parameters. (Refer to "Specifying Parameters" in chapter 4.)
- In addition to separating two numbers for calculations, the [ENTER] key has a few other uses. Refer to "Other Uses of the [ENTER] Key" on page 47.
- Pressing a key during a PSE (*pause*) causes program execution to stop. Press [R/S] to restart the program.

No Packing

If you're familiar with the HP-41, then you probably have seen the "PACKING" and "TRY AGAIN" messages. Packing removes any unused gaps in program memory. The HP-42S continuously keeps memory packed, so there is no need for a PACK function, and you'll never see a "PACKING" message.

Function Names

A number of function names used by the HP-42S are different from those on the HP-41, even though the functions work identically.

When keying in an HP-41 program, you can use *either* name for the functions in the following table. The calculator automatically converts each HP-41 function name to the corresponding HP-42S function. Note that HP-41 function names do not appear in the function catalog.

HP-41 Function Name	HP-42S Function Name
CHS	+/−
DEC	→DEC
D-R	→RAD
ENTER↑	ENTER
FACT	N!
FRC	FP
HMS	→HMS
HR	→HR
INT	IP
OCT	→OCT
P-R	→REC
RDN	R↓
R-D	→DEG
R-P	→POL

HP-41 Function Name	HP-42S Function Name
ST+	STO+
ST−	STO−
ST*	STO×
ST/	STO÷
X<=0?	X≤0?
X<=Y?	X≤Y?
*	×
/	÷

Stack Registers. The HP-42S distinguishes stack registers with `ST`. For example, the HP-41 instruction `10 VIEW X` is equivalent to the HP-42S instruction `10 VIEW ST X`.*

Alpha Strings. The HP-41 displays Alpha strings in programs with the ᵀ character. The HP-42S, however, surrounds Alpha strings with quotation marks. For example, the HP-41 program line `03 ᵀHELLO` is equivalent to the HP-42S instruction `03 "HELLO"`. Similarly, `04 ᵀ├THERE` is equivalent to `04 ├"THERE"`. (Note that some printers may not be able to print the append character.)

Example: Keying in an HP-41 Program. The following program was taken unaltered from the *HP-41CV Owner's Manual*. The program finds the roots of the equation $ax^2 + bx + c = 0$, where a, b, and c are constants. The solutions can be found using the quadratic formula, namely:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

* It is *not* equivalent to the instruction `10 VIEW "X"`, which displays a variable named X.

Key the program into memory:

HP-41 Program Listing:	HP-42S Keystrokes:
	[GTO] [.] [.] [PRGM]
01 LBL "QUAD"	[PGM.FCN] [LBL] QUAD [ENTER]
02 "a=?"	[ALPHA] a=?
03 PROMPT	[PGM.FCN] [▼] [PROM]
04 2	2
05 *	[×]
06 STO 00	[STO] 00
07 "b=?"	[ALPHA] b=?
08 PROMPT	[PGM.FCN] [▼] [PROM]
09 CHS	[+/−]
10 STO 01	[STO] 01
11 "c=?"	[ALPHA] c=?
12 PROMPT	[PGM.FCN] [▼] [PROM]
13 RCL 00	[RCL] 00
14 *	[×]
15 2	2
16 *	[×]
17 RCL 01	[RCL] 01
18 X↑2	[x^2]
19 X<>Y	[$x \gtrless y$]
20 -	[−]
21 X<0?	[PGM.FCN] [▼] [X?0] [X<0?]
22 GTO 01	[GTO] 01
23 SQRT	[$\sqrt{x}$]
24 STO 02	[STO] 02
25 RCL 01	[RCL] 01
26 +	[+]
27 RCL 00	[RCL] 00
28 /	[÷]
29 "ROOTS="	[ALPHA] ROOTS=
30 ARCL X	[ARCL] [.] [ST X]
31 AVIEW	[PGM.FCN] [AVIEW]
32 PSE	[PGM.FCN] [▼] [PSE]
33 RCL 01	[RCL] 01
34 RCL 02	[RCL] 02

35 -	[−]
36 RCL 00	[RCL] 00
37 /	[÷]
38 "AND "	[ALPHA] AND (*space*)
39 ARCL X	[ARCL] [·] ST X
40 AVIEW	[PGM.FCN] [PGM.FCN] AVIEW
41 RTN	RTN
42 LBL 01	LBL 01
43 "ROOTS COMPLEX"	[ALPHA] ROOTS COMPLEX [ENTER]
44 AVIEW	AVIEW
45 .END.	[EXIT]

After keying in the program, exit Program-entry mode and run the program for $a = 1$, $b = 7$, and $c = 12$.

[EXIT] [XEQ] QUAD

```
a=?
x: 0.0000
```

1 [R/S]

```
b=?
x: 2.0000
```

7 [R/S]

```
c=?
x: -7.0000
```

12 [R/S]

```
ROOTS=-3.0000
x: -3.0000
```

```
AND -4.0000
x: -4.0000
```

Enhancing HP-41 Programs

The HP-42S has a number of functions that you may want to incorporate into existing HP-41 programs. The following list can help you start thinking about enhancements for your HP-41 programs:

- Use named variables instead of storage registers to make your programs easier to understand (chapter 3).
- Take advantage of automatic labeling by using the INPUT and VIEW functions (chapter 9).
- Create CUSTOM menu key assignments that aid in executing programs or routines within programs (pages 68 and 112).
- Modify messages to take advantage of the larger display (page 129).
- Use program-controlled menus to enhance the *user interface* of a program (pages 125 and 145).

The *HP-42S Programming Examples and Techniques* manual (part number 00042-90020) uses the "QUAD" program from the preceding example to demonstrate how to enhance an HP-41 program.

Part 3

Built-In Applications

Page 178 12: The Solver

196 13: Numerical Integration

205 14: Matrix Operations

228 15: Statistics

245 16: Base Operations

12

The Solver

The built-in Solver application (■ SOLVER) is a special root finder that enables you to solve an equation for any of its variables. In this chapter, you'll learn how to:

- Solve for an unknown.
- Find the root(s) of an equation.
- Make initial guesses to help guide the Solver to a solution.
- Interpret the results returned by the Solver.
- Use the Solver in a program.

Additional examples using the Solver are included at the end of this chapter. They include the equation of motion for free-fall and the time value of money equation.

Using the Solver

The general procedure for solving is:

1. Enter a program that defines the function to be solved.
2. Press ■ SOLVER and then select the program you want to solve.
3. For each known variable, key in a value and then store the value by pressing the corresponding menu key.
4. Calculate the unknown variable by pressing the corresponding variable menu key.

Step 1: Writing a Program for the Solver

Before you use the Solver you must write a program or subroutine that evaluates *f(x)* for the function you want to solve. When you're writing the program, keep in mind that:

- The program must begin with a global label.
- The program must define the variables that will appear in the Solver variable menu.
- The Solver may execute your program many times to find a solution. Therefore, the length and efficiency of your program may affect the amount of time required to find a solution.

How the Solver Uses Your Program. The Solver executes your program using different values for the unknown variable. During each successive evaluation, the Solver moves closer to a solution. In most cases, the Solver eventually finds a value for the unknown variable that causes your function to evaluate to zero. This value is a solution.

Generally, the Solver finds a solution. However, it may encounter mathematical conditions in which a solution cannot be found. Refer to "How the Solver Works" on page 186.

Simplifying the Function. As with many mathematical procedures, the first step to solving a problem is simplification. Here you'll have to call upon your own expertise to simplify the equation. In general, you should attempt to combine like terms and constants, reducing the equation to the form

$$f(x) = 0$$

where *f(x)* is a function of one or more variables. For example, the equation for the volume of a box is given by

$$\text{Length} \times \text{Width} \times \text{Height} = \text{Volume}.$$

Rearranging terms gives

$$\text{Length} \times \text{Width} \times \text{Height} - \text{Volume} = 0.$$

Written as a program for the Solver, the function looks like this:

```
01 LBL "VOL"
02 MVAR "L"
03 MVAR "W"
04 MVAR "H"
05 MVAR "V"
06 RCL "L"
07 RCL× "W"
08 RCL× "H"
09 RCL- "V"
10 END
```

The global label identifies the program.

These lines identify the menu variables to appear in the Solver menu.

This is the body of the program that calculates *f(x)*. (Recalling data and recall arithmetic are covered in chapter 3.)

Defining Menu Variables. MVAR (*menu variable*) instructions define which variables appear in the Solver variable menu. These definitions must be grouped together (sequential line numbers) and must immediately follow the global label. The calculator ignores MVAR instructions that occur anywhere else in the program.

Your program may use any number of variables; however, only those defined with MVAR appear in the Solver variable menu.

The Body of the Program. The main purpose of the program is to calculate the function, *f(x)*. Key in the instructions just as if you were solving the equation from the keyboard. Recall each variable as it is needed.

Example: Keying In a Solver Program. Key the "VOL" program into your calculator.

A helpful hint: programs that use variables are easier to key in if the variables already exist. Before keying in the program, create the variables *V*, *H*, *W*, and *L* by storing a zero into each one.

0 [STO] [ENTER] V [ENTER]

```
Y: 0.0000
X: 0.0000
```

[STO] [ENTER] H [ENTER]

```
Y: 0.0000
X: 0.0000
```

STO ENTER W ENTER

```
Y: 0.0000
X: 0.0000
```

STO ENTER L ENTER

```
Y: 0.0000
X: 0.0000
```

Go to a new program space, select Program-entry mode, and key in the "VOL" program listed above.

■ GTO . .
■ PRGM

```
00▶{ 0-Byte Prgm }
01 .END.
```

■ PGM.FCN LBL VOL ENTER

```
00 { 7-Byte Prgm }
01▶LBL "VOL"
```

Pressing ■ SOLVER in Program-entry mode displays a menu containing the MVAR function.

■ SOLVER MVAR L

```
02▶MVAR "L"
MVAR          PSLV SOLVE
```

MVAR W

```
03▶MVAR "W"
MVAR          PSLV SOLVE
```

MVAR H

```
04▶MVAR "H"
MVAR          PSLV SOLVE
```

MVAR V EXIT

```
04 MVAR "H"
05▶MVAR "V"
```

RCL L

```
05 MVAR "V"
06▶RCL "L"
```

RCL × W

```
06 RCL "L"
07▶RCL× "W"
```

RCL × H

```
07 RCL× "W"
08▶RCL× "H"
```

RCL − V

```
08 RCL× "H"
09▶RCL- "V"
```

Press EXIT to exit Program-entry mode.

Step 2: Selecting a Program To Solve

When you execute the Solver from the keyboard (■ SOLVER), it prompts you to select a program. All global labels that are followed by MVAR instructions are displayed in a menu. Select a program by pressing the corresponding menu key. (If there are more than six labels, use ▲ or ▼ to find the program you're looking for.)

Example. Select the "VOL" program entered in the previous example. The Solver immediately displays the variable menu for "VOL".

■ SOLVER VOL

```
x: 0.0000
 L  W  H  V
```

Step 3: Storing the Known Variables

When you select a program to solve, the calculator searches for menu variables used by the program and displays a variable menu. Use the variable menu to store values into the known variables. Refer to page 125 for more information about using variable menus.

Example. Store these dimensions: *length* = 5 cm, *width* = 7 cm, and *height* = 12 cm. Key in each value and then press the corresponding menu key.

5 L

```
L=5.0000
 L  W  H  V
```

7 W

```
W=7.0000
 L  W  H  V
```

12 H

```
H=12.0000
 L  W  H  V
```

Step 4: Solving for the Unknown

After storing the known values, all that remains is to press the menu key for the unknown. The Solver immediately begins searching for a solution. During this process, the Solver displays two numbers. These numbers represent the two current estimates of the solution.

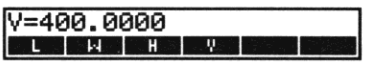
Example. Solve for the volume of a box using the dimensions entered in the previous example.

V

The volume is 420 cm^3.

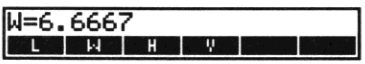
Using the same length and height, what is the width of a box if the volume is 400 cm^3? Store the known volume.

400 V



Solve for the width.

W



Choosing Initial Guesses

By entering guesses, you can control the initial estimates used in a search for a solution. Since the search starts in the range between the two initial estimates, entering guesses can reduce the number of iterations required to find a solution. Also, if more than one solution exists, guesses can help select the solution you desire.

A useful application of providing initial guesses is finding multiple roots of an equation. For example, the expression $(x - 3)(x - 2)$ has roots at $x = 3$ and $x = 2$. The root that the Solver finds depends on the starting point for its search. Initial guesses tell the Solver where to begin.

To enter guesses for the unknown variable:

1. Key in the first guess; press the menu key for the unknown variable.
2. Key in the second guess; press the menu key again.
3. Press the menu key a third time to begin solving.

Example: Finding Multiple Roots of an Equation. A solution for a single unknown, say x, is a root if $f(x) = 0$. Consider the following equation:

$$x^3 - 5x^2 - 10x = -20.$$

Rearranging terms gives

$$x^3 - 5x^2 - 10x + 20 = 0.$$

By factoring out an x, the equation is easier to write as a program.

$$x(x^2 - 5x - 10) + 20 = 0$$

Key in the following program:

```
01 LBL "FNX"
02 MVAR "X"
03 RCL "X"
04 ENTER
05 X↑2
06 LASTX
07 5
08 ×
09 -
10 10
11 -
12 ×
13 20
14 +
15 END
```

The program defines a single menu variable, X.

Recalls X and makes an extra copy.

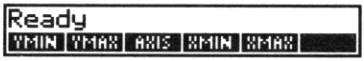
Calculates $(x^2 - 5x - 10)$.

Calculates $x(x^2 - 5x - 10)$ using the extra copy of X made in line 04.

Completes $f(x) = x(x^2 - 5x - 10) + 20$.

If you have the "DPLOT" program in your calculator (page 156), you can plot $f(x) = x^3 - 5x^2 - 10x + 20$ in the display like this:

XEQ DPLOT



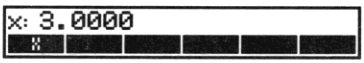
50 +/- YMIN 25 YMAX

0 AXIS

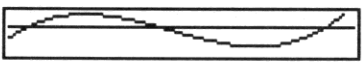
3 +/- XMIN 7 XMAX

R/S

FNX R/S

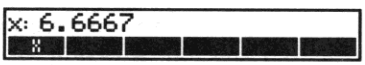


X



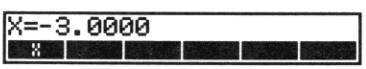
By examining the plot, you can see that there are three roots (intersections with the x-axis). Use the Solver to find each root.

SOLVER FNX



Since X is the only variable declared in the program, it's the only one that appears in the Solver menu. By carefully choosing your guesses, you can zero in on each root. The graph shows that the first root is somewhere between $x = -3$ and $x = 0$. Enter the first guess.

3 +/- X

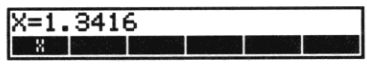


Enter the second guess and then solve for X.

0 [X] [X]

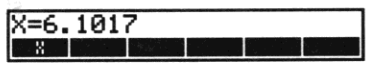
The first root is $x = -2.4433$. Now use the same procedure to find the second root, which from the graph appears to be between $x = 0$ and $x = 4$.

0 [X] 4 [X] [X]



The second root is $x = 1.3416$. Calculate the third root, which appears to be between $x = 4$ and $x = 7$.

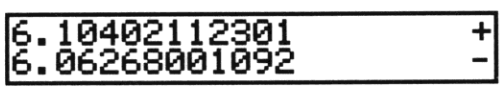
4 [X] 7 [X] [X]



The third root is $x = 6.1017$.

How the Solver Works

The Solver uses an iterative (repetitive) process to search for a solution that sets the function equal to zero. The Solver starts with two initial estimates of the answer—your guesses, or numbers it generates. Using one of the estimates, the Solver evaluates your program. Then, the Solver repeats the calculation using the other estimate. If neither estimate produces a value of zero, the Solver produces two new estimates that appear to be closer to the answer. By repeating this process many times, the Solver approaches a solution.



During the search for a solution, the calculator displays the two current estimates for the unknown.* Next to each estimate, the calculator displays a sign (+ or −). Each sign indicates whether the function is positive or negative at that estimate.

* Estimates are not displayed when a program executes the Solver.

A question mark next to an estimate indicates the function cannot be evaluated at that estimate. Generally, this is because of some mathematical error, such as dividing by zero.

Halting and Restarting the Solver

Depending on the function you are solving, it can take several minutes to find a solution. You can halt the search by pressing [R/S] (or [EXIT]). To resume the search from where it left off, press [R/S] again.

If the estimates don't seem to be proceeding towards a number you judge to be a reasonable answer, halt the search (press [R/S]), and then enter new guesses and start over.

Interpreting the Results

There are several possible outcomes of an iterative search for a solution. The Solver returns data to the stack registers that can be used to help you interpret the results. For a more detailed description of these conditions, refer to the *HP-42S Programming Examples and Techniques* manual (part number 00042–90020).

Stack Register	Contents
T	An integer (0–4) indicating the condition that caused the Solver to stop. 0 = A solution has been found. 1 = A sign reversal has occurred. 2 = An extremum has been found. 3 = Bad guess(es) were used. 4 = The function may be a constant.
Z	The value of the function evaluated at the solution. If an actual root has been found, the Z-register contains a zero.
Y	The previous guess.
X	The solution (or the best guess if a solution was not found).

Solution Found. A solution has been found that may be a root. If you want to know if the result is an actual root, you can:

- Test the contents of the Z-register. If this number is equal to zero, then the solution is an actual root.
- Press the menu key to solve for the unknown again. If you get the same result (without a message), the solution is an actual root. If, on the other hand, you see the message `Sign Reversal`, then the result is only an approximation to a root.

Sign Reversal. A discontinuity or pole has been found. The Solver has found neighboring points for which the value of the function changes sign, but no point at which it evaluates to zero.

Extremum. The Solver has found an approximation to a local minimum or maximum of the numerical absolute value of the function. If the solution is $\pm 9.99999999999 \times 10^{499}$, it corresponds to an asymptotic extremum.

Bad Guess(es). If the Solver stops and displays `Bad Guess(es)`, one or both initial guesses lie outside of the domain of the function. That is, the function returns an error when evaluated at the values of the guesses.

Constant? If the Solver stops and displays `Constant?`, the function returns the same value at every point sampled by the Solver, suggesting that the function may be constant.

Using the Solver in a Program

To use the Solver in a program, the program must:

1. Select a program using the PGMSLV (*program to solve*) function.
2. Store the known variables.
3. Provide initial guesses for the unknown (optional). The first guess is stored into the variable. The second guess is taken from the X-register.
4. Solve for the unknown with the SOLVE function.

For example, the following program segment illustrates how the "VOL" program could be solved by another program. This program multiplies the current value of *L* by 3 and stores that value into *H*. That value is then multiplied by 3 again and stored into *V*. The program then solves for *W*.

```
01 LBL "BOXSLV"
02 PGMSLV "VOL"
03 RCL "L"
04 3
05 ×
06 STO "H"
07 3
08 ×
09 STO "V"
10 SOLVE "W"
11 GTO IND ST T
      ⋮
```

Selects "VOL" as the program to solve.

Calculates new values for *H* and *V*.

Solves for *W*.

Branches to the subroutine specified by the code (0–4) in the T-register. That is, the program branches to LBL 00 if a solution is found, LBL 01 if a sign reversal occurred, LBL 02 if an extremum was found, LBL 03 if the guesses were bad, or LBL 04 if the function is a constant. (Refer to the table on page 187).

More Solver Examples

The Equation of Motion for Free-Fall

The equation of motion for a free-falling object is

$$Distance = v_0t + \frac{1}{2}gt^2$$

where v_0 is the initial velocity, t is the time, and g is the acceleration due to gravity. The Solver enables you to solve for any of the variables, given values for the other variables.

Rearranging terms gives

$$0 = v_0t + \frac{1}{2}gt^2 - Distance.$$

Written as a program for the Solver, the equation looks like this:

```
01 LBL "FREE"
02 MVAR "Dist"
03 MVAR "Vo"
04 MVAR "Time"
05 MVAR "g"
```

Defines the menu variables for the program.

```
06 RCL "Vo"
07 RCL "Time"
08 ×
```

Calculates v_0t.

```
09 LASTX
10 X↑2
11 RCL× "g"
12 2
13 ÷
```

Calculates $\frac{1}{2}gt^2$.

```
14 +
```

Adds the two intermediate results: $v_0t + \frac{1}{2}gt^2$.

```
15 RCL- "Dist"
16 END
```

Subtracts the distance, which completes $f(x)$.

Since the acceleration due to gravity, g, is a menu variable, you can change it to match the units of the problem you're working. It also allows you to calculate g based on experimental data.

Example. Calculate how far an object falls in 5 seconds (starting from rest). Before you begin, go to a new program space and key in the program listed above.

SOLVER FREE

The object is starting at rest, so $v_0 = 0$.

0 VO

Store the appropriate acceleration constant. To get a final result in meters, use 9.8 m/s^2.

9.8 G

Store the time (5 seconds).

5 TIME

Now solve for the distance.

DIST

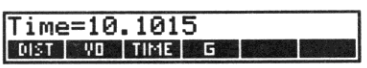
An object falls 122.5 meters in 5 seconds.

Try another calculation: how long does it take an object to fall 500 meters? Since v_0 and g are already stored, there's no need to store them again. Store the distance.

500 DIST

Calculate the time.

TIME



It takes slightly more than 10 seconds for an object to fall 500 meters.

The Time Value of Money Equation

The time value of money equation

$$0 = PV + (1 + ip)\ PMT \left[1 - \frac{(1 + i)^{-N}}{i}\right] + FV\ (1 + i)^{-N}$$

establishes the relationships between the following variables:

N	The number of monthly payments or compounding periods.
I%YR	The annual interest rate as a fraction ($i = I\%YR \div 1200$).
PV	The present value. (This can also be an initial cash flow or a discounted value of a series of future cash flows.) *PV* always occurs at the beginning of the first month.
PMT	The monthly payment.
FV	The future value. (This can also be a final cash flow or a compounded value of a series of cash flows.) *FV* always occurs at the end of the *N*th month.

The value *p* indicates payment timing. If $p = 1$, then payments occur at the *beginning* of each month. If $p = 0$, then payments occur at the *end* of each month. The "TVM" program uses flag 00 to represent *p*. For payments at the beginning a each month, set flag 00. For payments at the end of each month, clear flag 00.

Here is how the equation can be written as a program for the Solver:

```
01 LBL "TVM"
02 MVAR "N"
03 MVAR "I%YR"
04 MVAR "PV"
05 MVAR "PMT"
06 MVAR "FV"
```

Declares the menu variables.

```
07 1
08 ENTER
09 ENTER
10 RCL "I%YR"
11 %
12 12
13 ÷
14 STO ST T
```

Calculates the monthly interest rate expressed as a decimal fraction, i.

```
15 FC? 00
16 CLX
17 +
```

If flag 00 is clear (End mode), calculates $(i + 0)$. If flag 00 is set (Begin mode), calculates $(i + 1)$.

```
18 R↓
19 +
20 RCL "N"
21 +/-
22 Y↑X
```

Calculates $(1 + i)^{-N}$.

```
23 1
24 X<>Y
25 -
```

Calculates $1 - (1 + i)^{-N}$.

```
26 LASTX
27 RCL× "FV"
```

Calculates $FV\ (1 + i)^{-N}$.

```
28 R↓
29 X<>Y
30 ÷
```

Calculates $1 - \frac{(1 + i)^{-N}}{i}$.

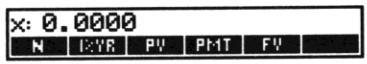
```
31 ×
32 RCL× "PMT"
33 +
34 RCL+ "PV"
35 END
```

Completes the expression.

Example. Penny of Penny's Accounting wants to know what the monthly payments will be for a 3-year loan at 10.5% annual interest, compounded monthly. The amount financed is $5,750. Payments are made at the end of each period.

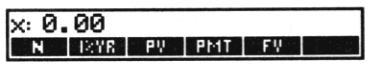
After keying in the program above, use the Solver to calculate the unknown information for Penny.

[SOLVER] TVM



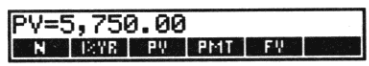
Clear flag 00 and set the display format to FIX 2.

[FLAGS] CF 00
[DISP] FIX 02



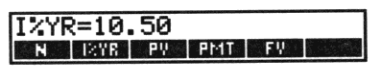
Enter the known values: $PV = 5750$, $FV = 0$, $I\%YR = 10.5$, and $N = 3 \times 12$.

5750 PV

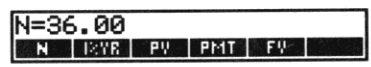


0 FV

10.5 I%YR

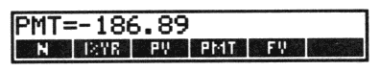


3 [ENTER] 12 [×] N



Now solve for the payment.

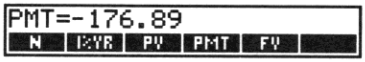
PMT



The payment is negative because it is money to be *paid out*.

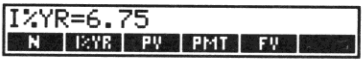
This is $10 higher than Penny's client can pay each month. What interest rate would reduce the monthly payments by $10? *Add* 10 to the negative payment that's already in the X-register and store the new value into *PMT*.

10 + PMT



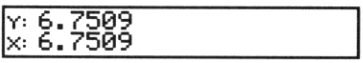
Now, solve for the interest rate.

I%YR



Return to FIX 4 display mode and exit from the Solver.

■ DISP FIX 04 EXIT EXIT

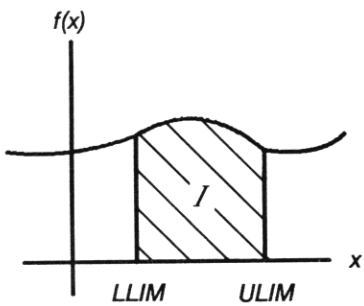


13

Numerical Integration

Many problems in mathematics, science, and engineering require calculating the definite integral of a function. If the function is denoted by *f(x)* and the interval of integration is from the lower limit (*LLIM*) to the upper limit (*ULIM*), then the integral can be expressed mathematically as

$$I = \int_{LLIM}^{ULIM} f(x)\, dx.$$



The quantity *I* can be interpreted geometrically as the area of a region bounded by the graph of *f(x)*, the *x*-axis, and the limits $x = LLIM$ and $x = ULIM$ (provided that *f(x)* is nonnegative throughout the interval of integration).

In this chapter, you'll learn how to use the HP-42S Integration application (■ ∫f(x)) to calculate a definite integral.