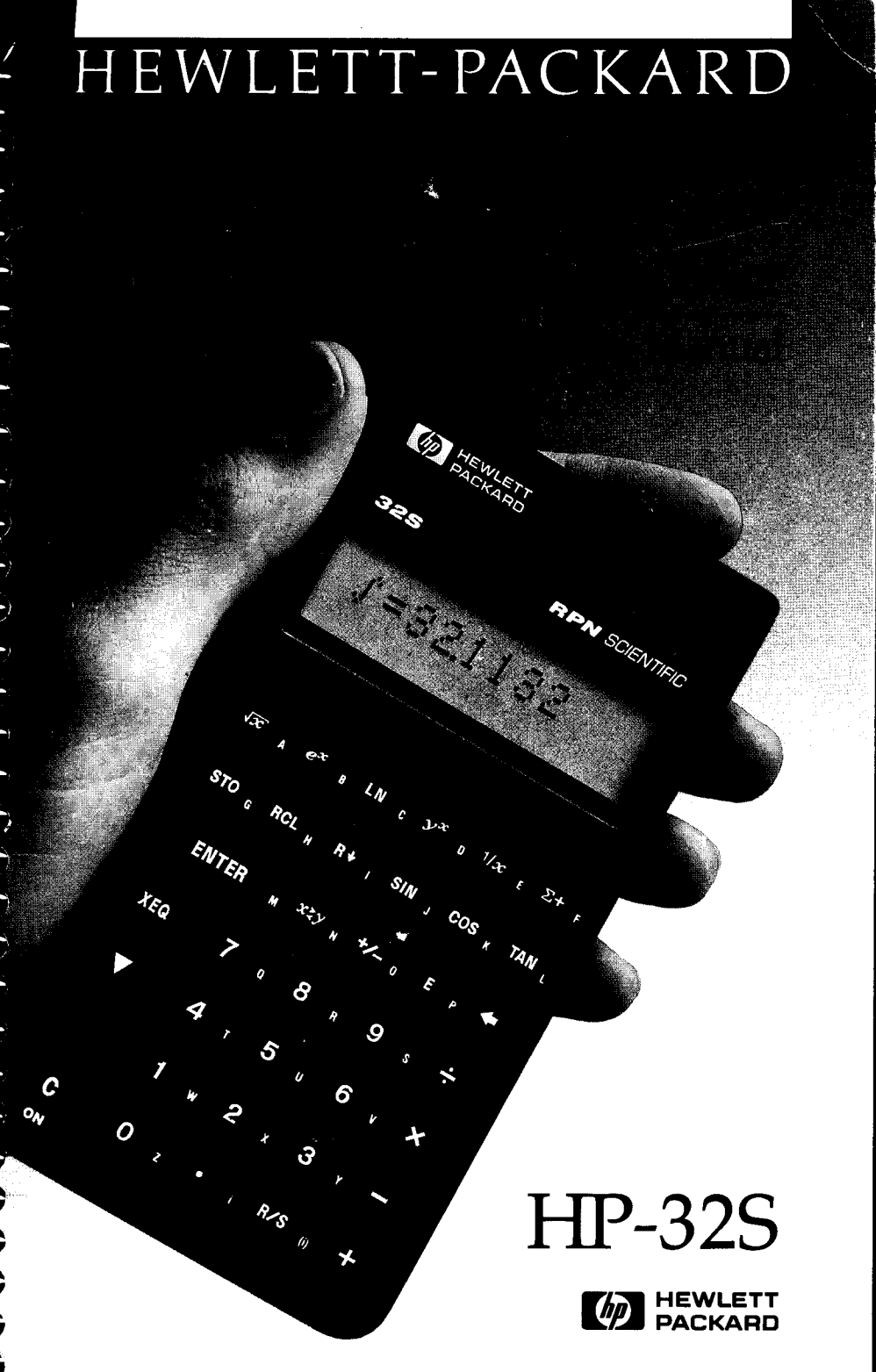
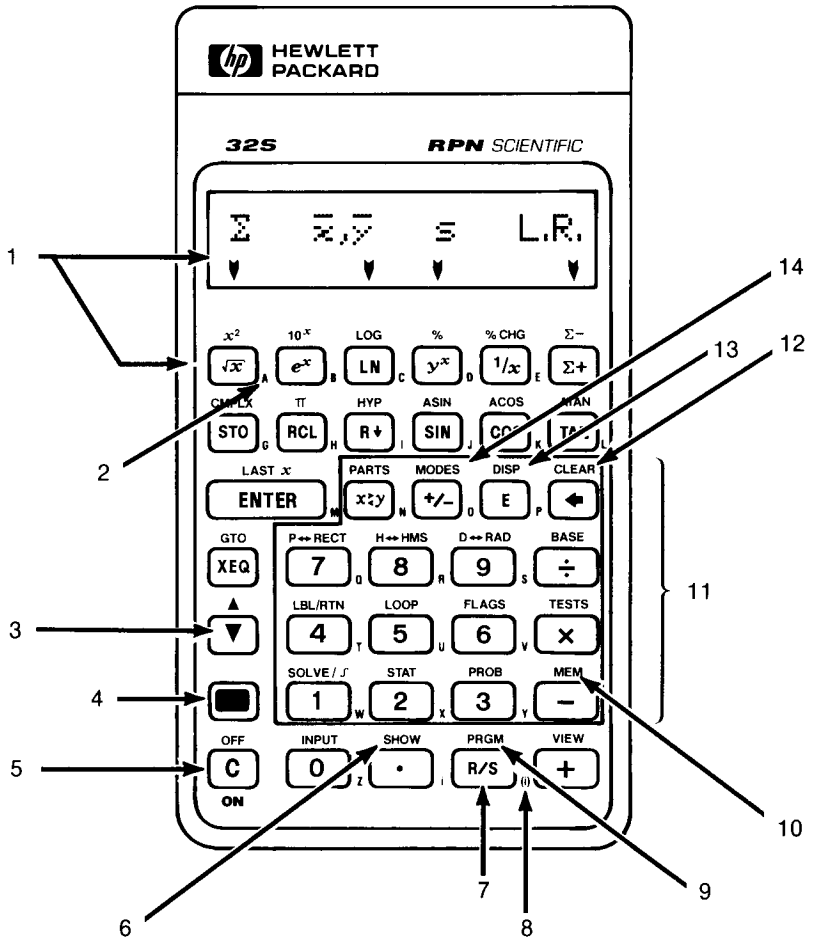
HEWLETT-PACKARD



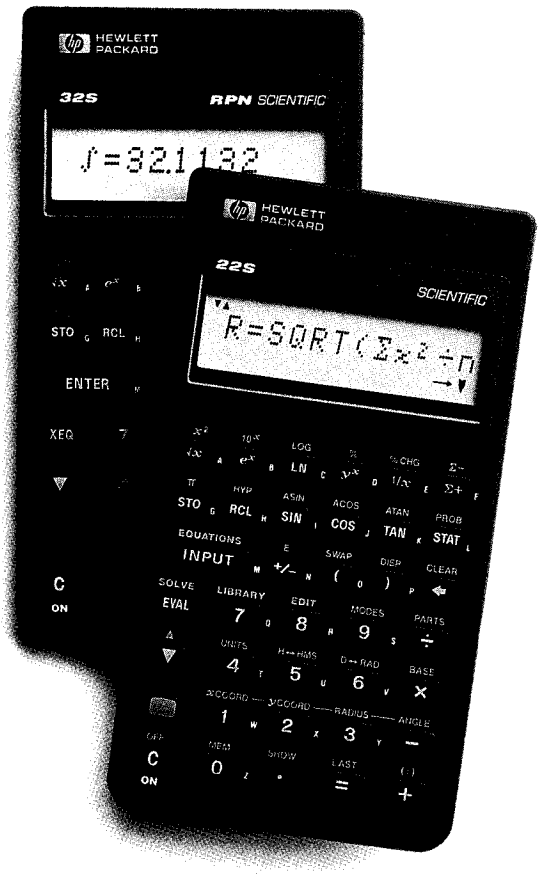
HP-32S

HEWLETT
PACKARD



1. Menu and menu keys.
2. Letter keys for variables & labels.
3. Steps through programs and lists.
4. Shift key.
5. On; cancel display, menu, program entry.
6. Shows all decimal places.
7. Run/Stop toggle for programs.
8. For indirect addressing via *i*.
9. Toggles in and out of program entry.
10. User memory; stored variables and programs.
11. Menu keys (boxed area)
12. Clears all or parts of memory.
13. Display formats.
14. Angular modes, periods & commas.

The Hewlett-Packard calculator you've just purchased will serve you faithfully for many years. Whether you're a student or a seasoned professional, your HP calculator will help you solve the tough problems your work demands.



The Perfect Complement to a New HP Calculator

HP's Step-by-Step solutions books are the closest thing to having a custom calculator at your fingertips. These handy books offer a variety of examples and keystroke procedures to help you set up your calculations the way you need them.

Specific to a wide range of topics within science and engineering, these books will help you with precisely the problems you need to solve.

For the HP-22S:

Science Student Applications $9.95
(00022-90034)

- Solve mathematical problems in algebra, trigonometry, linear algebra, and calculus.
- Learn methods for solving problems in physics, chemistry, thermodynamics, statics and dynamics, and electrical fundamentals.
- Learn advanced equation-writing techniques.

For the HP-32S:

Engineering Applications $9.95
(00032-90057)

- Solve problems in electrical engineering: reactance chart, impedance of a ladder network, Smith chart conversions, transistor amplifier performance.
- Perform mechanical engineering calculations: black body radiation, conduit flow, composite section properties, and Soderberg's equation.
- Calculate civil engineering problems including Mohr's circle for stress and field angle traverse.
- Perform statistics calculations of Chi-square, t statistics, F distribution, and analysis of variance.
- Solve math problems: triangle solutions and linear interpolation.

Elegant Leather Cases

Protect your new HP-22S or HP-32S in style with a handsome leather case.

Black (HP 92169K)	$19.00
Brown (HP 92169L)	$19.00
Burgundy (HP 92169M)	$19.00

Owner's Manuals

Each HP calculator comes with an owner's manual. Additional manuals may be ordered separately as well.

Manuals for HP calculators are also available in a variety of languages. Contact your HP dealer or local HP sales office for more information.

For More Information

For additional information on calculator accessories and a demonstration of Hewlett-Packard professional calculators or handheld computers, visit your nearest HP dealer. For the location and number of the dealer nearest you, call toll-free 1-800-752-0900.

How To Order

To order items your local dealer does not carry, call toll-free 1-800-538-8787. Please refer to Call Code P180 when ordering. MasterCard, Visa and American Express cards are welcome.

All prices are suggested U.S. list and are subject to change without notice.

HEWLETT PACKARD

HP-32S
RPN Scientific Calculator

Owner's Manual

HEWLETT PACKARD

Edition 2 September 1988
Reorder Number 00032-90039

Notice

For warranty and regulatory information, see pages 248 and 252.

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

Corvallis Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Printing History

Edition 1	March 1988	Mfg. No. 00032-90040
Edition 2	September 1988	Mfg. No. 00032-90065

Welcome to the HP-32S

Your HP–32S reflects the superior quality and attention to detail in engineering and manufacturing that have distinguished Hewlett-Packard products for more than 40 years. Hewlett-Packard stands behind this calculator: we offer accessories, worldwide service, and expertise to support its use (see inside the back cover).

Hewlett-Packard Quality

Our calculators are made to excel, to last, and to be easy to use.

- This calculator is designed to withstand the usual drops, vibrations, pollutants (smog, ozone), temperature extremes, and humidity variations that it may encounter in normal, everyday worklife.
- The calculator and its manual have been designed and tested for ease of use. We selected spiral binding to let the manual stay open to any page, and we added many examples to highlight the varied uses of this calculator.
- Advanced materials and permanent, molded-in key lettering provide a long keyboard life and a positive feel to the keyboard.
- CMOS (low-power) electronics and the liquid-crystal display allow data to be retained even when the calculator is off, and let the batteries last a long time.
- The microprocessor has been optimized for fast and reliable computations using 15 digits internally for precise results.
- Extensive research has created a design that has minimized the adverse effects of static electricity (a potential cause of malfunctions and data loss in calculators).

Features

The feature set of this calculator reflects needs and wishes we solicited from customers. The HP-32S features:

- All functions available either on the keyboard or in menus: you do not have to type their names in.
- Messages and program lines in English, such as `DIVIDE BY 0` instead of `ERR 21`. Data storage in variables *A* through Z.
- Our traditional RPN logic, which saves keystrokes.
- 390 bytes of memory to store data and programs.
- Advanced functionality for statistics, base conversions, complex-number arithmetic, integration, and solving for the unknown variable of an equation.
- Extensive HP programming capability, including editing, labeled input and output, subroutines, looping, conditional instructions, flags, and indirect addressing.

Contents

Part 1: Basic Operation

Part 2: Programming

Part 3: Advanced Operation

Part 4: Application Programs

Part 5: Appendixes and Reference

Part 1

Basic Operation

1

Getting Started

Important Preliminaries

Turning the Calculator On and Off

To turn the calculator on, press [C]. Note ON printed below the key.

To turn the calculator off, press [■] [OFF]. That is, press and release the shift key ([■]), then press [C] (which has OFF printed above it). Since the calculator has *Continuous Memory*, turning it off does not affect any information you've stored.

To conserve energy, the calculator turns itself off after about 10 minutes of no use.

Under most conditions, the calculator's batteries last well over a year. If you see the low-power indicator () in the display, replace the batteries as soon as possible. See appendix A for details and instructions.

Adjusting the Display's Contrast

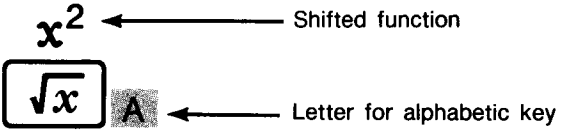
The display's brightness depends on lighting, your viewing angle, and the contrast setting. To darken or lighten the display, hold down the [C] key and press [+] or [−].

Highlights of the Keyboard and Display

Shifted Keystrokes

Each key has two functions: one printed on its face and a *shifted* function printed in color above the key. Press the colored shift key (■) before these functions. For example, to turn the calculator off, press and release ■, then press [C]. This is written as ■[OFF].

Pressing ■ turns on the shift annunciator (⬏), which remains until you press the next key. To cancel ⬏, just press ■ again.



The Letter Keys

Most of the keys have a letter written next to them, as shown above. Whenever you need to type in a letter—which is used to identify a variable or a label—the **A..Z** annunciator appears in the display, indicating that the letter keys are "active". (Variables are covered in chapter 3.)

Backspacing and Clearing

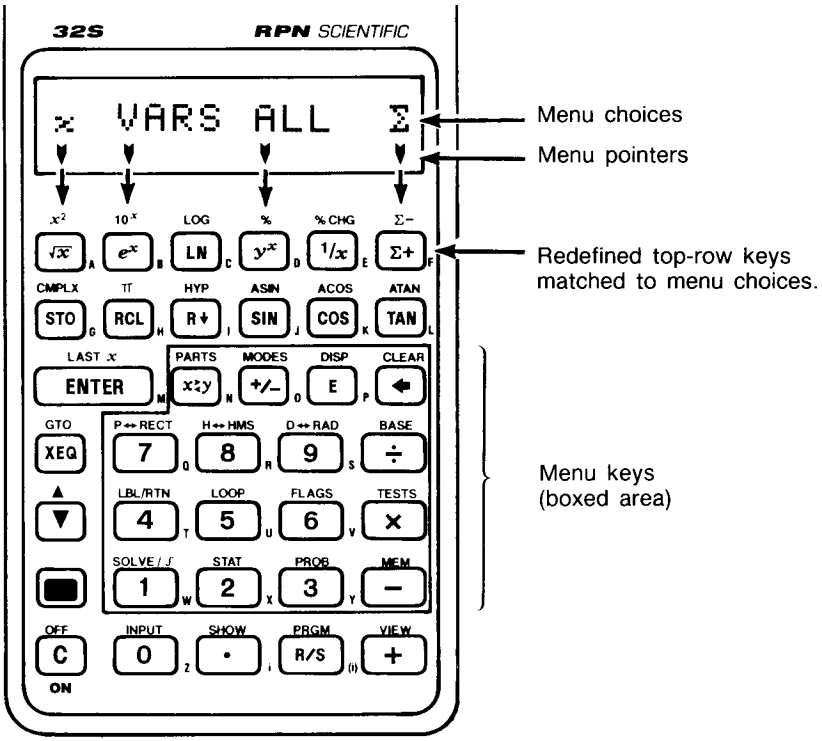
One of the first things you need to know is how to *clear:* how to correct numbers, clear the display, and, in general, start over.

Keys for Clearing

Key	Description
◀	*Backspace.* Erases the last character before the cursor (_) or backs out of the current menu. For a completed number (no cursor), ◀ clears the entire number. Also clears error messages. During program entry: deletes the program line.
C	*Clear* or *Cancel. Clears* the displayed number to zero or *cancels* the current situation (such as a menu, a message, a prompt for input, a catalog, or program entry).
■ CLEAR	*The CLEAR menu.* Gives you options for clearing data: {x}, {VARS}, {ALL}, and {Σ}. These clear: the current number (called "x"), all variables, all of memory, and statistical data. During program entry, the menu includes {PGM}, which erases all of program memory.

Using Menus

There is a lot more power to the HP-32S than what you see printed on the keyboard. This is because almost half of the shifted keys are *menu keys,* which, when pressed, offer you several more functions—or more options for more functions. This extra power is easier to find than if each function had its own key.



Those shifted functions printed with lighter backgrounds on the calculator (such as **CLEAR**) are *menu keys*. Pressing a menu key produces a *menu* in the display—a series of choices.

HP-32S Menus

Menu	Description	In Chapter:
	Numeric Functions	
PARTS	Number-altering functions (integer part, absolute value, etc.).	4
P↔RECT	Conversions between polar and rectangular coordinates.	4
H↔HMS	Conversions between hours and hours-minutes-seconds.	4
D↔RAD	Conversions between degrees and radians.	4
BASE	Base conversions.	10
SOLVE/∫	Functions for root-solving and integration.	7, 8
STAT	Statistical functions.	11
PROB	Probability functions.	4
	Programming Instructions	
LBL/RTN	Label, return (end), and pause.	5
LOOP	Conditional looping and counting functions.	6
FLAGS	Functions to set, clear, and test flags.	6
TESTS	Conditional tests.	6
	Other Functions	
MODES	Angular modes and decimal-point convention.	4, 1
DISP	Display formats.	1
CLEAR	Functions to clear data.	1, 3, 5
MEM	Status of memory: memory used for individual variables and programs. Catalogs for variables and programs.	1

For example, to find the factorial of 25:

Keys:	**Display:**	**Description:**
25	25_	Displays number.
[■] [PROB]	Cn,r Pn,r x! R	Displays the Probability menu.
{x!} (the [1/x] key)	1.5511E25	25! is 1.5511×10^{25}.

In this way, menus help you execute dozens of functions by guiding you to them with menu choices. There is no need to remember the exact names of all of the many functions available on the HP-32S nor to search through many names printed on the keyboard.

Exiting Menus

Whenever you execute a function in a menu, the menu automatically disappears, as in the example above. If you wish to leave a menu *without* executing a function, you have three options:

- Pressing [←] **backs out of the menu**, one step at a time.

123	123_
[■] [PROB]	Cn,r Pn,r x! R
{R}	RANDOM SEED
[←]	Cn,r Pn,r x! R
[←]	123.0000

- Pressing [C] **cancels the menu.**

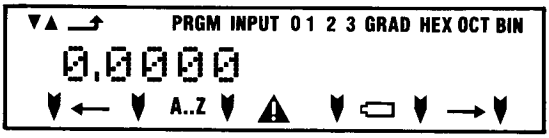
123	123_
[■] [PROB]	Cn,r Pn,r x! R
{R}	RANDOM SEED
[C]	123.0000

■ **Pressing any other menu key** replaces the old menu with the new one.

123	123_
■ [PROB]	Cn,r Pn,r x! R
{R}	RANDOM SEED
■ [CLEAR]	x VARS ALL Σ

Annunciators

The symbols shown here are called *annunciators.* Each one has a special significance when it appears in the display.



Annunciator	Meaning
▼▲	[▼] and ■[▲] are active for stepping through a program or a list (pages 33, 76).
⎯↗	Shift (■) is active (page 15).
PRGM	Program entry is active (pages 72, 75).
INPUT	Program is waiting for input; enter number and press [R/S] to resume the program (page 77).
0 1 2 3	Specifies which flags are set (page 98).
RAD GRAD	Radians or Grads angular mode is set (page 57).
HEX OCT BIN	Specifies which number base is active (page 144).
▼	Top-row keys are redefined according to the menu labels above the menu pointers (page 17).

Annunciator	Meaning
←, →	There are more digits to the left or right. Use ■ [SHOW] to see the rest of a decimal number; use left and right scrolling keys ([√x], [Σ+]) to see the rest of a binary number (page 150).
A..Z	The alphabetic keys are active (page 48).
⚠	Attention! Indicates a special condition or an error (pages 21, 32).
▭	Battery power is low (page 242).

Keying In Numbers

You can key in a number that has up to 12 digits plus a 3-digit exponent up to ±499. If you try to key in a number larger than this, digit entry halts and the ⚠ annunciator briefly appears.

If you make a mistake while keying in a number, press [←] to backspace and delete the last digit, or press [C] to clear the whole number.

Making Numbers Negative

The [+/−] key changes the sign of a number.

- To key in a negative number, type that number, then press [+/−].
- To change the sign of a number that was entered previously, just press [+/−]. (If the number has an exponent, [+/−] affects only the *mantissa*—the *non*-exponent part.)

Exponents of Ten

Exponents in the Display. Numbers with exponents of ten (such as 4.2×10^{-5}) are shown in the display with an E preceding the exponent (such as 4.2000E-5). A number whose magnitude is too large or too small for the display format will automatically be displayed in exponential form. For example, in FIX 4 format for four decimal places, observe the effect of the following keystrokes:

Keys:	Display:	Description:
.000062	0.000062_	Shows number being entered.
[ENTER]	0.0001	Rounds number to fit display format.
.000042 [ENTER]	4.2000E-5	Automatically uses scientific notation because otherwise no significant digits would appear.

Keying In Exponents of Ten. Use [E] (*exponent*) to key in numbers multiplied by powers of ten. For example, take Planck's constant, 6.6262×10^{-34}.

1. Key in the *mantissa* (the non-exponent part) of the number. If this part is negative, press [+/-].

   6.6262 → 6.6262_

2. Press [E]. Notice that the cursor moves behind the E.

   [E] → 6.6262E_

3. Key in the exponent. (Largest possible exponent is ±499.) If it is negative, press [+/-].

   34 [+/-] → 6.6262E-34_

For a power of ten without a multiplier, such as 10^{34}, just press [E] 34. The calculator displays 1E34.

Other Exponent Functions. To *specify* an exponent *of ten* while entering a number, use [E]. To *calculate* an exponent of ten (the base 10 antilogarithm), use [■] [10^x] (chapter 4). To *calculate* the result of *any* number raised to a power (exponentiation), use [y^x] (chapter 4).

Understanding Digit Entry

As you key in a number, the *cursor* (_) appears in the display. The cursor shows you where the next digit will go; it therefore indicates that this number is not completed yet. In technical talk, we say that *digit entry is not terminated.*

Keys:	Display:	Description:
123	123_	Digit entry is not terminated: the number is not completed.

If you *execute a function* to calculate a *result,* then the cursor disappears because the number is complete. Digit entry has been terminated.

[√x]	11.0905	Digit entry terminated.

Pressing [ENTER] also terminates digit entry. This is why you must separate two numbers with [ENTER]: to terminate one number before starting to key in the second one.

123 [ENTER]	123.0000	A completed number.
5 [+]	128.0000	Another completed number.

If digit entry is not terminated (the cursor is present), then [←] backspaces to erase the last digit. If digit entry is terminated (no cursor), then [←] acts like [C] and clears the entire number. Try it!

Range of Numbers and OVERFLOW

The smallest magnitude of a number available on the calculator is 1×10^{-499}. The largest magnitude is $9.99999999999 \times 10^{499}$ (displayed as `1.0000E500` because of rounding).

- If a calculation produces a result that exceeds the largest possible magnitude, then the number $9.99999999999 \times 10^{499}$ is provided instead. The warning message `OVERFLOW` appears.
- If a calculation produces a result smaller than the smallest possible magnitude, then zero is provided instead. There is no warning.

Doing Arithmetic

When you press a function key, the calculator immediately executes the function written on that key. Therefore, all operands (numbers) must be present *before* you press the function key.

All calculations can be broken down into one-number functions and two-number functions.

One-Number Functions

To use a one-number function (such as [1/x], [√x], [■][x²], and [+/−]):

1. Key in the number. (*You do not need to press* [ENTER].)
2. Press the function key. (For a shifted function, press the shift key first.)

For example, calculate 1/32 and $\sqrt{148.84}$. Then square the last result and change its sign.

Keys:	Display:	Description:
32	`32_`	Operand.
[1/x]	`0.0313`	Reciprocal.
148.84 [√x]	`12.2000`	Square root.

[■] [x²]	148.8400	Square of 12.2.
[+/−]	−148.8400	Negation of 148.84.

The one-number functions also include the trigonometric functions, the logarithmic functions, the hyperbolic functions, and the parts-of-numbers functions, all of which are discussed in chapter 4.

Two-Number Functions

To use a two-number function (such as [+], [−], [×], and [÷]):

1. Key in the first number.
2. Press [ENTER] to separate the first number from the second.
3. Key in the second number. (*Do not press* [ENTER].)
4. Press the function key. (For a shifted function, press the shift key first.)

Remember to enter both numbers before executing the function.

For example:

To Calculate:	Press:	Display Is:
12 + 3	12 [ENTER] 3 [+]	15.0000
12 − 3	12 [ENTER] 3 [−]	9.0000
12 × 3	12 [ENTER] 3 [×]	36.0000
12 ÷ 3	12 [ENTER] 3 [÷]	4.0000

The order of entry is, of course, essential for noncommutative functions such as [−] and [÷]. If the numbers have been entered in the wrong order, you can still get the correct answer without re-entering the numbers by pressing [x≷y] to *swap the order of the numbers.* Then perform the intended function. (This is explained in detail in chapter 2 under "Stack Manipulations.")

Chain Calculations

The speed and simplicity of calculating with the HP-32S are apparent during *chain calculations* (that is, calculations with more than one operation). Even during the longest of calculations, *you still work with only one or two numbers at a time*—the automatic memory stack stores intermediate results until you need them, then it inserts them into the calculation.*

- This method requires fewer keystrokes than other calculator logic does, and it adapts itself naturally to programming.
- The process of working through a problem is the same as working it out on paper, but the calculator does the hard part.

For example, solve (12 + 3) × 7.

Work From the Parentheses Out. If you were working this problem out on paper, you would first calculate the intermediate result of (12 + 3)...

$$\overset{15}{(\cancel{12 + 3})} \times 7 =$$

...and then you would multiply the intermediate result by 7.

$$15 \times 7 = 105$$

Solve the problem in the same way on the HP-32S, starting inside the parentheses.

Keys:	Display:	Description:
12 [ENTER] 3 [+]	15.0000	Calculates the intermediate result first.

You don't need to press [ENTER] to save this intermediate result before proceeding. Since it is a calculated result, it is saved automatically.

* Don't worry now about the *automatic (RPN) memory stack* and how it works. The stack is explained in chapter 2.

7 [×]	105.0000	Pressing the function key produces the answer. This result can be used in further calculations.

Now study these examples. Notice that you only press [ENTER] to separate *sequentially entered* numbers, such as at the beginning of a problem. The operations themselves ([+], [−], etc.) separate subsequent numbers and save intermediate results. The last result saved is the first one retrieved as needed to carry out the calculation.

First calculate $\frac{2 + 3}{10}$:

Keys:	**Display:**	**Description:**
2 [ENTER] 3 [+] 10 [÷]	0.5000	(2 + 3) ÷ 10.

Now calculate $\frac{2}{3 + 10}$:

Keys:	**Display:**	**Description:**
3 [ENTER] 10 [+]	13.0000	Calculates (3 + 10) first.
2 [x≷y] [÷]	0.1538	Puts 2 *before* 13 so the division is correct: 2 ÷ 13.

Calculate $\frac{14 + 7 + 3 - 2}{4}$.

Keys:	**Display:**	**Description:**
14 [ENTER] 7 [+] 3 [+] 2 [−]	22.0000	Calculates (14 + 7 + 3 − 2) first.
4 [÷]	5.5000	22 ÷ 4.

Now calculate $\frac{4}{14 + (7 \times 3) - 2}$.

7 [ENTER] 3 [×]	21.0000	Calculates (7 × 3).
14 [+] 2 [−]	33.0000	Calculates bracketed numbers next.
4 [x≷y]	33.0000	Puts 4 *before* 33 in preparation for division.
[÷]	0.1212	Calculates 4 ÷ 33, the answer.

Problems that have multiple parentheses can be solved in the same simple manner, using the automatic storage of intermediate results. For example, to solve (3 + 4) × (5 + 6) on paper, you would

first calculate the quantity inside these parentheses...

$$(3 + 4) \times (5 + 6)$$

...and then the quantity inside these parentheses...

...and then you would multiply the two intermediate answers together.

You work through the problem the same way with the HP-32S, except that you don't have to write down intermediate answers—the calculator remembers them for you.

Keys:	**Display:**	**Description:**
3 [ENTER] 4 [+]	7.0000	First adds (3 + 4).
5 [ENTER] 6 [+]	11.0000	Then adds (5 + 6).
[×]	77.0000	Then multiplies the intermediate answers together for the final answer.

Remember. This method of entering numbers, called Reverse Polish Notation, is unambiguous and therefore does not need parentheses.

- You never work with more than two numbers at a time.
- Use [ENTER] to separate two numbers keyed in sequentially.
- Pressing a function key immediately executes that function.
- Intermediate results appear as they are calculated, so you can check each step as you go.
- Intermediate results are automatically stored. They reappear automatically as they are needed for the calculation—the last result stored is the first to come back out.
- You can calculate in the same order as you would with pencil and paper.

Exercises

Calculate: $\frac{\sqrt{(16.3805 \times 5)}}{0.05} = 181.000$

Solution: 16.3805 [ENTER] 5 [×] [√x] .05 [÷]

Calculate: $\sqrt{[(2 + 3) \times (4 + 5)]} + \sqrt{[(6 + 7) \times (8 + 9)]} = 21.5743$

Solution: 2 [ENTER] 3 [+] 4 [ENTER] 5 [+] [×] [√x] 6 [ENTER] 7 [+] 8 [ENTER] 9 [+] [×] [√x] [+]

Calculate: $(10 - 5) \div [(17 - 12) \times 4] = 0.2500$

Solution: 17 [ENTER] 12 [−] 4 [×] 10 [ENTER] 5 [−] [x≷y] [÷]

or

10 [ENTER] 5 [−] 17 [ENTER] 12 [−] 4 [×] [÷]

Controlling the Display Format

Periods and Commas in Numbers

To exchange the periods and commas used for the decimal point (radix mark) and digit separators in a number:

1. Press ■ [MODES] to display the MODES menu.

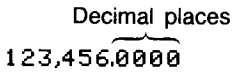
2. Specify the decimal point by pressing {.} or {,}. For example, the number one million looks like:
   - 1,000,000.0000 if you press {.}, or
   - 1.000.000,0000 if you press {,}.

Number of Decimal Places (■ DISP)

All numbers are *stored* with 12-digit precision,* but you can select the number of decimal places to be *displayed* by using the ■ DISP (*display*) function. The displayed number is *rounded* according to the display format. The DISP menu gives you four options:

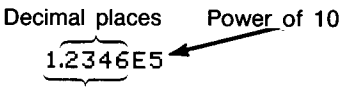
```
FX SC EN ALL
```

Fixed-Decimal Format ({FX}). FIX format displays a number with up to 11 decimal places (if they fit). After the prompt FIX _, specify the number of decimal places to be displayed. For 10 or 11 places, press ⊡ 0 or ⊡ 1.



Any number that is too large or too small to display in the current setting will automatically be displayed in scientific format.

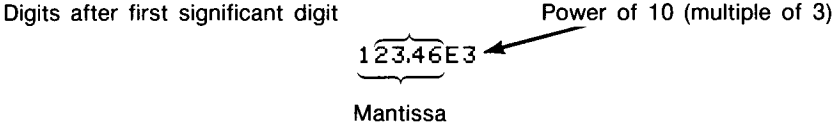
Scientific Format ({SC}). SCI format displays a number in scientific notation (one number before the decimal point) with up to 11 decimal places (if they fit) and up to three digits in the exponent. After the prompt, SCI _, specify the number of decimal places. (The integer part will always be less than 10.) For 10 or 11 places, press ⊡ 0 or ⊡ 1.



* During some complicated internal calculations, the calculator uses 15-digit precision for intermediate results.

Engineering Format ({EN}). ENG format displays a number in a manner similar to scientific notation, but the exponent is a multiple of three (and therefore there can be one, two, or three digits before the decimal point). This is most useful for scientific and engineering calculations that use units specified in multiples of 10^3 (such as micro-, milli-, and kilo-units).

After the prompt, ENG _, specify the number of digits you want *after* the first significant digit. (The integer part will always be less than 1,000.) For 10 or 11 digits, press [·] 0 or [·] 1.

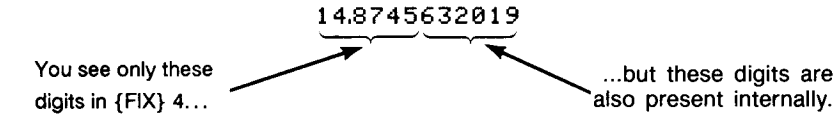


ALL Format ({ALL}). ALL format displays a number as precisely as possible (12 digits maximum). If not all digits fit in the display, the number is automatically displayed in scientific format.

123,456

SHOWing Full 12-Digit Precision

Changing the number of displayed decimal places affects what you see, but it does not affect the internal representation of numbers. The number stored internally always has 12 digits.



To temporarily display a number with its full precision, press ■ [SHOW]. This shows you *just the mantissa* (no exponent) of the number for as long as you hold down [SHOW].

Keys:	Display:	Description:
[■] [DISP] {FX} 4		Displays four decimal places.
45 [ENTER] 1.3 [×]	58.5000	Four decimal places displayed.
[■] [DISP] {SC} 2	5.85E1	Scientific format: two decimal places and an exponent.
[■] [DISP] {ALL}	58.5	All significant digits; trailing zeros dropped.
[■] [DISP] {FX} 4	58.5000	Four decimal places, no exponent.
[1/x]	0.0171	
[■] [SHOW] (hold)	170940170940	Temporarily shows full precision.

Messages

The calculator responds to certain conditions or keystrokes by displaying a message. The ⚠ symbol comes on to call your attention to the message.

- To clear a message, press [C] or [◀].
- To clear the message *and* perform another function, press any other key.

If no message appears but ⚠ does, then you have pressed an *inactive key* (a key that has no meaning in the current situation, such as the [3] key in Binary mode).

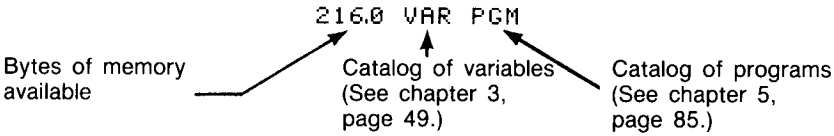
All displayed messages are explained in the list of messages on page 281.

Calculator Memory

There are 390 bytes of user memory in the HP-32S available to you for any combination of stored data (variables or program lines). The memory requirements of specific activities are given under "Managing Calculator Memory" in appendix B.

Checking Available Memory

Pressing ■ [MEM] shows you the amount of memory still available:



1. To enter the catalog of variables, press {VAR}. To enter the catalog of programs, press {PGM}.
2. To review the catalogs, press [▼] or ■ [▲].
3. To delete a variable or a program, press ■ [CLEAR] while viewing it in its catalog.
4. To exit the catalog, press [C].

Clearing All of Memory

Clearing all of memory erases all numbers and program lines you've stored. It does not affect settings (modes and formats). (To clear settings as well as data, see "Clearing Memory" in appendix B.)

To clear all of memory:

1. Press ■ CLEAR {ALL}. You will then see the confirmation prompt CLR ALL? Y N, which safeguards against the unintentional clearing of memory.
2. Press {Y} (*yes*).

2

The Automatic Memory Stack

This chapter explains how calculations take place in the automatic memory stack and why this method minimizes the number of keystrokes for complicated calculations. *You do not need to read and understand this material to use the calculator.* However, you will find that understanding this material greatly enhances your use of the calculator, especially when programming.

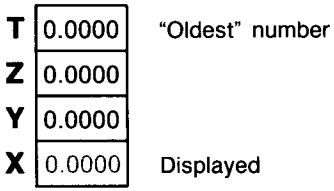
In part 2, "Programming", you will see that the stack helps manipulate and organize data for programs.

What the Stack Is

Automatic storage of intermediate results is the reason that the HP-32S easily processes the most complex calculations, and does so without parentheses. The key to automatic storage is the *automatic, RPN memory stack.**

The memory stack consists of four storage locations, called *registers,* which are "stacked" on top of each other. It is a work area for calculations. These registers—labeled X, Y, Z, and T—store and manipulate four current numbers. The "oldest" number is the one in the T- (*top*) register.

* HP's operating logic is based on an unambiguous, parentheses-free mathematical logic known as "Polish Notation," developed by the Polish logician Jan Łukasiewicz (1878—1956). While conventional algebraic notation places the operators *between* the relevant numbers or variables, Łukasiewicz's notation places them *before* the numbers or variables. For optimal efficiency of the stack, we have modified that notation to specify the operators *after* the numbers. Hence the term *Reverse Polish Notation,* or *RPN.*



The most "recent" number is in the X-register: *this is the number you see in the display.*

In programming, the stack is used to perform calculations, to temporarily store intermediate results, to pass stored data (variables) among programs and subroutines, to accept input, and to deliver output.

The X-Register Is in the Display. The X-register is what you see *except* when a menu, a message, or a program line is being displayed. You might have noticed that several functions' names include an *x* or *y*. This is no coincidence: these letters refer to the X- and Y-registers. For example, ■ [10^x] raises ten to the power of the number in the X-register (the displayed number).

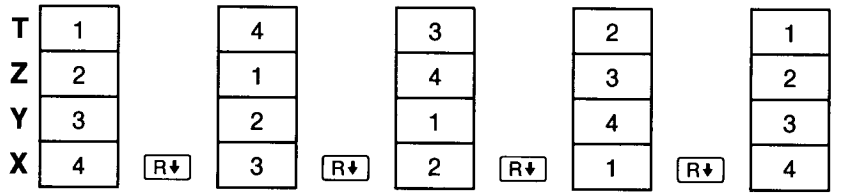
■ [CLEAR] {x} **versus** [C]. Pressing ■ [CLEAR] {x} *always* clears the X-register to zero, and it is also used to program this instruction. The [C] key, in contrast, is context-sensitive. It either clears or cancels the current display, depending on the situation: it acts like ■ [CLEAR] {x} only when the X-register is displayed.* It *cancels* other displays: menus, labeled numbers, messages, and program entry.

Reviewing the Stack ([R↓])

The [R↓] (*roll down*) key lets you review the entire contents of the stack by "rolling" the contents downward, one register at a time. You can see each number when it enters the X-register.

* [◆] also acts like ■ [CLEAR] {x} when the X-register is displayed *and* digit entry is terminated (no cursor present).

Suppose the stack is filled with 1, 2, 3, 4 (press 1 [ENTER] 2 [ENTER] 3 [ENTER] 4). Pressing [R↓] four times rolls the numbers all the way around and back to where they started:



What was in the X-register rotates around and enters the T-register. Notice that the *contents* of the registers are rolled. The registers themselves maintain their positions, and the X-register is always displayed.

Exchanging the X- and Y-Registers in the Stack ([x≷y])

Another key that manipulates the stack contents is [x≷y] (*x exchange y*). It swaps the contents of the X- and Y-registers without affecting the rest of the stack. Pressing [x≷y] twice, of course, restores the original order of the contents.

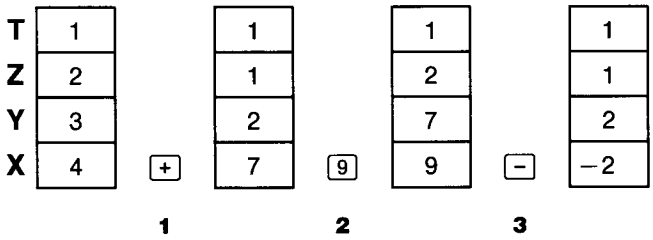
The [x≷y] function is used primarily for two purposes:

- To view y and then return it to the Y-register (press [x≷y] twice). Some functions yield two results: one into the X-register and one into the Y-register. An example is {y,x→θ,r}, which converts rectangular coordinates in the X- and Y-registers into polar coordinates in the X- and Y-registers.
- To swap the order of numbers in a calculation. For example, an easy way to calculate 9 ÷ (13 × 8) is to press 13 [ENTER] 8 [×] 9 [x≷y] [÷].

Arithmetic—How the Stack Does It

The contents of the stack move up and down automatically as new numbers enter the X-register (*lifting the stack*) and as operators combine two numbers to produce one new number (*dropping the stack*) in the X-register. Suppose the stack is still filled with the numbers 1, 2, 3, 4. See how the stack drops and lifts its contents while calculating

$$3 + 4 - 9:$$



1 The stack "drops" its contents. (The top register replicates its contents.)

2 The stack "lifts" its contents. (The top contents are "lost".)

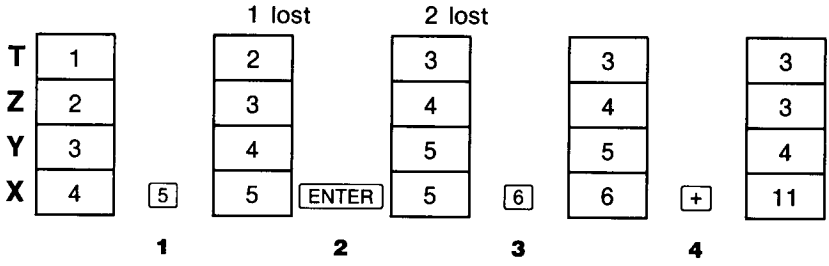
3 The stack drops.

- Notice that when the stack lifts, it pushes the top contents out of the T-register, and that number is lost. You can see, therefore, that the stack's memory is limited to four numbers for calculations.
- When the stack drops, it replicates the contents of the T-register.
- Because of the automatic movement of the stack, you do *not* need to clear the display before doing a new calculation.
- Most functions prepare the stack to lift its contents *when the next number enters the X-register*. See appendix B for lists of functions that affect stack lift.

How ENTER Works

You know that [ENTER] separates two numbers keyed in one after the other. In terms of the stack, how does it do this? Suppose the stack is again filled with 1, 2, 3, and 4. Now enter and add two new numbers:

5 + 6:



1 Lifts the stack.

2 Lifts the stack and replicates the X-register.

3 Does *not* lift the stack.

4 Drops the stack and replicates the T-register.

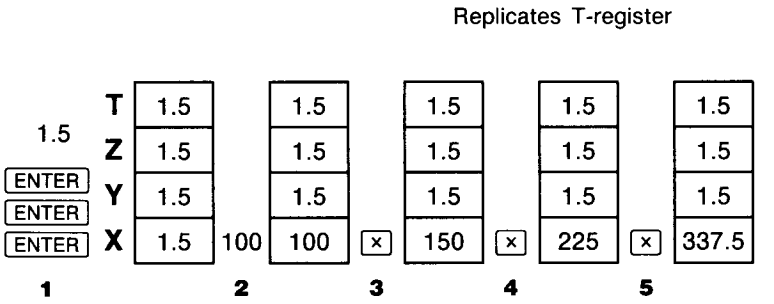
[ENTER] replicates the contents of the X-register into the Y-register. The next number you key in (or recall) *writes over* the copy of the first number left in the X-register. The effect is simply to separate two sequentially entered numbers.

You can use the replicating effect of [ENTER] to clear the stack quickly: press 0 [ENTER] [ENTER] [ENTER]. All registers now contain zero. Note, however, that you don't *need* to clear the stack before doing calculations.

Using a Number Twice in a Row. You can use the replicating feature of [ENTER] to other advantages. To add a number to itself, press [ENTER] [+].

Filling the Stack With a Constant. The replicating effect of [ENTER] together with the replicating effect (from T into Z) of stack drop allows you to fill the stack with a numeric constant for calculations.

Example: Constant, Cumulative Growth. Given a bacterial culture with a constant growth rate of 50%, how large would a population of 100 be at the end of 3 days?



1 Fills the stack with the growth rate.

2 Keys in the initial population.

3 Calculates the population after 1 day.

4 Calculates the population after 2 days.

5 Calculates the population after 3 days.

How CLEAR x Works

Clearing the display (X-register) puts a zero in the X-register. The next number you key in (or recall) *writes over* this zero.

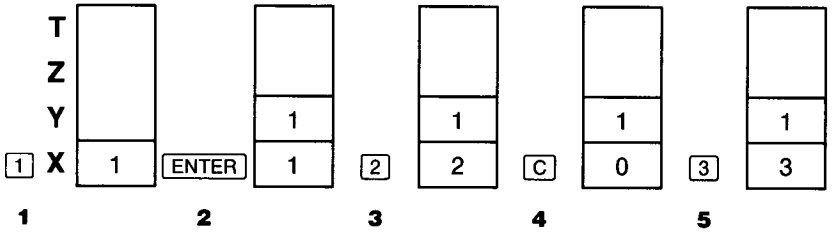
There are three ways to clear the number in the X-register, that is, to *clear x*:

- Press [C].
- Press [←].
- Press [f] [CLEAR] {x}. (These keystrokes are mainly used in program entry.)

Note these exceptions:

- During program entry, [←] deletes program lines and [C] cancels program entry.
- During digit entry, [←] backspaces over the number.
- If the display shows a *labeled* number (such as A=2.0000), pressing [C] or [←] cancels that display and shows the X-register.

For example, if you wanted to enter 1 and 3 but mistakenly entered 1 and 2, this is what you would do to correct it:



1 Lifts the stack.

2 Lifts the stack and replicates the X-register.

3 Overwrites the X-register.

4 Clears *x* by overwriting it with zero.

5 Overwrites *x* (replaces the zero.)

The LAST X Register

The LAST X register is a companion to the stack: it holds the number that was in the X-register before the last numeric function was executed. (A numeric function is an operator that produces a result from another number or numbers, such as [√x].) Pressing ■ [LASTx] returns this value into the X-register. This ability to retrieve the "last *x*" has two main uses: correcting errors and reusing a number in a calculation.

See appendix B for a comprehensive list of the functions that save *x* in the LAST X register.

Correcting Mistakes With ■ LASTx

Wrong One-Number Function. If you execute the wrong one-number function, use ■ LASTx to retrieve the number so you can execute the correct function. (Press C *first* if you want to clear the incorrect result from the stack.)

Since ■ % and ■ %CHG don't cause the stack to drop, you can recover from these functions in the same manner as from one-number functions.

Example. Suppose that you had just computed $\ln 4.7839 \times (3.879 \times 10^5)$ and wanted to find its square root, but pressed e^x by mistake. You don't have to start over! To find the correct result, just press ■ LASTx $\sqrt{x}$.

Mistakes With Two-Number Functions. If you make a mistake with a two-number operation (+, −, ×, ÷, or y^x), you can correct it by using ■ LASTx and the inverse of the two-number function (− or +, ÷ or ×, 1/x or y^x):

1. Press ■ LASTx to recover the second number (x just before the operation).
2. Execute the inverse operation. This returns the number that was originally first. The second number is still in the LAST X register. Then:
   - If you had used the *wrong function,* press ■ LASTx again to restore the original stack contents. Now execute the correct function.
   - If you had used the *wrong second number,* key in the correct one and execute the function.

If you had used the *wrong first number,* key in the correct first number, press ■ LASTx to recover the second number, and execute the function again. (Press C *first* if you want to clear the incorrect result from the stack.)

Example. Suppose you made a mistake while calculating

$$16 \times 19 = 304.$$

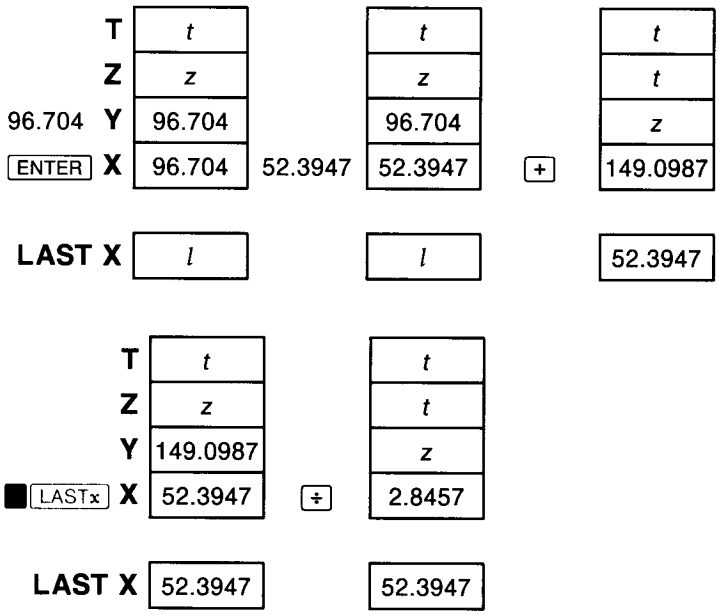
There are three kinds of mistakes you could have made:

Wrong Calculation	Mistake	Correction
16 [ENTER] 19 [−]	Wrong function.	[■] [LASTx] [+] [■] [LASTx] [×]
15 [ENTER] 19 [×]	Wrong first number.	16 [■] [LASTx] [×]
16 [ENTER] 18 [×]	Wrong second number.	[■] [LASTx] [÷] 19 [×]

Reusing Numbers With [■] [LASTx]

You can use [■] [LASTx] to reuse a number (such as a constant) in a calculation. Remember to enter the constant second, just before executing the arithmetic operation, so that the constant is the last number in the X-register, and therefore can be saved and retrieved with [■] [LASTx].

Example. Calculate $\dfrac{96.704 + 52.3947}{52.3947}$.



Keys:	Display:	Description:
96.704 [ENTER]	96.7040	
52.3947 [+]	149.0987	Intermediate result.
■ [LASTx]	52.3947	Brings back display from before [+].
[÷]	2.8457	Final result.

Example. Two close stellar neighbors of Earth are Rigel Centaurus (4.3 light-years away) and Sirius (8.7 light-years away). Use *c*, the speed of light (9.5×10^{15} meters per year) to convert the distances from the Earth to these stars into meters.

$$\text{to Rigel Centaurus} = 4.3 \text{ yr.} \times (9.5 \times 10^{15} \text{ m/yr.})$$
$$\text{to Sirius} = 8.7 \text{ yr.} \times (9.5 \times 10^{15} \text{ m/yr.})$$

Keys:	Display:	Description:
4.3 [ENTER]	4.3000	Light-years to R. Centaurus.
9.5 [E] 15	9.5E15_	Speed of light, *c*.
[x]	4.085E16	Distance to R. Centaurus.
8.7 ■ [LASTx]	9.5000E15	Retrieves *c*.
[x]	8.2650E16	Distance to Sirius.

Chain Calculations

The automatic lifting and dropping of the stack's contents let you retain intermediate results without storing or reentering them, and without using parentheses. This is an advantage the RPN stack has over other data-handling methods.

Order of Calculation

In chapter 1 we recommended solving chain calculations by working from the innermost parentheses outward. However, you can also choose to work problems in a left-to-right order.

For example, in chapter 1 you calculated:

$$4 \div [14 + (7 \times 3) - 2]$$

by starting with the innermost parentheses (7×3) and working outward—just as you would with pencil and paper. The keystrokes were:

7 [ENTER] 3 [×] 14 [+] 2 [−] 4 [x≷y] [÷].

Working the problem left-to-right, the solution would be:

4 [ENTER] 14 [ENTER] 7 [ENTER] 3 [×] [+] 2 [−] [÷],

which takes one additional keystroke. Notice that the first intermediate result is still the innermost parentheses: (7×3). The advantage to working a problem left-to-right is that you don't have to use [x≷y] to reposition operands for noncommutative functions ([−] and [÷]).

The first method (starting with the innermost parentheses) is often preferred because:

- It takes fewer keystrokes.
- It requires fewer registers in the stack.

When using a left-to-right method, be sure that no more than four intermediate numbers (or results) will be needed at one time, since the stack can hold no more than four numbers at once. This example, when solved left-to-right, needed all the registers in the stack at one point.

$$4 \div [14 + (7 \times 3) - 2]$$

Keys:	Display:	Description:
4 [ENTER] 14 [ENTER]	14.0000	Saves 4 and 14 as intermediate numbers in stack.
7 [ENTER] 3	3_	At this point the stack is full with numbers for this calculation.
[×]	21.0000	Intermediate result.
[+]	35.0000	Intermediate result.
2 [−]	33.0000	Intermediate result.
[÷]	0.1212	Final result.

Exercises

Here are some extra problems that you can do to practice using RPN.

Calculate: $(14 + 12) \times (18 - 12) \div (9 - 7) = 78.0000$
A Solution: 14 [ENTER] 12 [+] 18 [ENTER] 12 [−] [×] 9 [ENTER] 7 [−] [÷]

Calculate: $23^2 - (13 \times 9) + 1/7 = 412.1429$
A Solution: 23 [■] [x^2] 13 [ENTER] 9 [×] [−] 7 [1/x] [+]

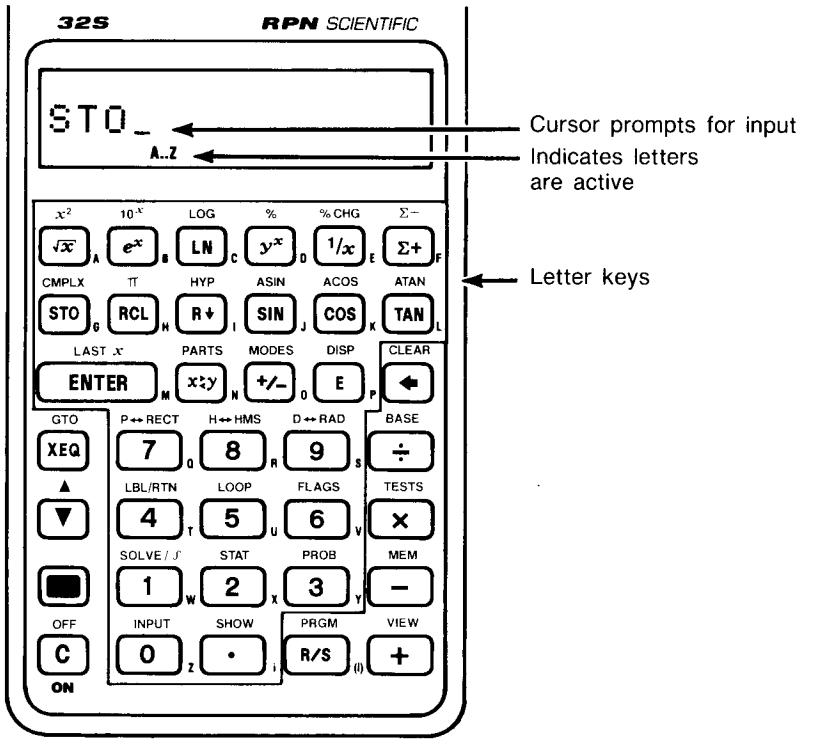
Calculate: $\sqrt{(5.4 \times 0.8) \div (12.5 - 0.7^3)} = 0.5961$
A Solution: 5.4 [ENTER] .8 [×] .7 [ENTER] 3 [y^x] 12.5 [x≷y] [−] [÷] [$\sqrt{x}$]
or
5.4 [ENTER] .8 [×] 12.5 [ENTER] .7 [ENTER] 3 [y^x] [−] [÷] [$\sqrt{x}$]

Calculate: $\sqrt{\dfrac{8.33 \times (4 - 5.2) \div [(8.33 - 7.46) \times 0.32]}{4.3 \times (3.15 - 2.75) - (1.71 \times 2.01)}} = 4.5728$
A Solution: 4 [ENTER] 5.2 [−] 8.33 [×] [■] [LASTx] 7.46 [−] 0.32 [×] [÷]
3.15 [ENTER] 2.75 [−] 4.3 [×] 1.71 [ENTER] 2.01 [×] [−] [÷] [$\sqrt{x}$]

3

Storing Data Into Variables

The HP-32S has 390 bytes of *user memory*: memory space that you can use to store numbers or program lines. Numbers are stored in locations called *variables*, each named with a letter from *A* through Z. (You can choose the letter to remind you of what is stored there, such as *B* for *bank balance* and *C* for the speed of light.)*



* Note that the variables *X*, *Y*, *Z*, and *T* are *different* storage locations from the X-register, Y-register, Z-register, and T-register in the stack.

Each white letter is associated with a key and a unique variable. The letter keys are automatically active when needed. (The **A..Z** annunciator in the display confirms this.)

Storing and Recalling Numbers

Numbers are stored into and recalled from lettered variables with the [STO] (*store*) and [RCL] (*recall*) functions.

To store a copy of a number from the display (X-register) to a variable: press [STO] *letter-key*.

To recall a copy of a number from a variable to the display: press [RCL] *letter-key*.

Example: Storing Numbers. Store Avogadro's number (approximately 6.0225×10^{23}) in *A*.

Keys:	**Display:**	**Description:**
6.0225 [E] 23	6.0225E23_	
[STO]	STO _	Prompts for variable.
A ([√x] key)	STO A	Displays function as long as key is held down.
	6.0225E23	Stores a copy of Avogadro's number in *A*. This also terminates digit entry (no cursor present).
[C]	0.0000	Clears the number in the display.
[RCL]	RCL _	Prompts you for the variable's name.
A	6.0225E23	Copies Avogadro's number from *A* to the display.

Viewing a Variable Without Recalling It. The ■ [VIEW] function shows you the contents of a variable without putting that number in the X-register. The display is labeled for the variable, such as:

```
A=1,234.5678
```

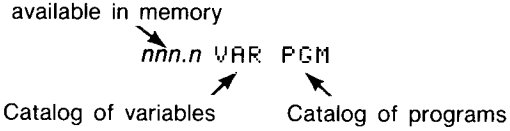
If the number is too large to fit completely in the display with its label, it is rounded and the rightmost digits are dropped. (An exponent is displayed in full.) To see the full mantissa, press ■ [SHOW].

■ [VIEW] is most often used in programming, but it is useful anytime you want to view a variable's value without affecting the contents of the stack.

To cancel the VIEW display, press [◄] or [C] once.

Reviewing Variables in the VAR Catalog

The ■ [MEM] (*memory*) function provides information about memory:



To review the values of any or all non-zero variables:

1. Press ■ [MEM] {VAR}.
2. Press [▼] or ■ [▲] to move the list and display the desired variable. (Note the ▼▲ annunciator, indicating that the arrow keys are active.)

   To see all the significant digits of a number displayed in the {VAR} catalog, press ■ [SHOW]. (If it is a binary number with more than 12 digits, use the [√x] and [Σ+] keys to see the rest.)
3. To copy a displayed variable from the catalog to the X-register, press [ENTER].
4. To clear a variable to zero, press ■ [CLEAR] while it is displayed in the catalog.
5. Press [C] to cancel the catalog (or [◄] to back up to the menu).

Clearing Variables

Variables' values are retained by Continuous Memory until you replace them or clear them. *Clearing* a variable stores a zero there; a value of zero takes no memory.

To clear a single variable: store zero in it.

To clear selected variables:

1. Press ■ MEM {VAR} and use ▼ or ■ ▲ to display the variable.
2. Press ■ CLEAR.
3. Press C to cancel the catalog, or ◄ to back out.

To clear all variables at once: press ■ CLEAR {VARS}.

Arithmetic With Stored Variables

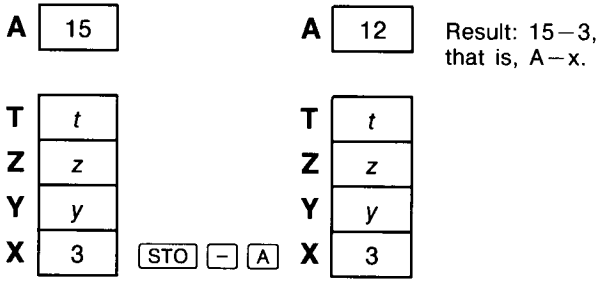
Storage arithmetic and *recall arithmetic* allow you to do calculations with a number stored in a variable *without recalling the variable into the stack.* A calculation uses one number from the X-register and one number from the specified variable.

Storage Arithmetic

Storage arithmetic uses STO +, STO −, STO ×, or STO ÷ to do arithmetic in the variable itself and to store the result there. It uses the value in the X-register and does not affect the stack.

New value of variable = Previous value of variable $\{+, -, \times, \div\}$ x

For example, suppose you want to reduce the value in A (15) by the number in the X-register (3, displayed). Press STO − A. Now A = 12, while 3 is still in the display.

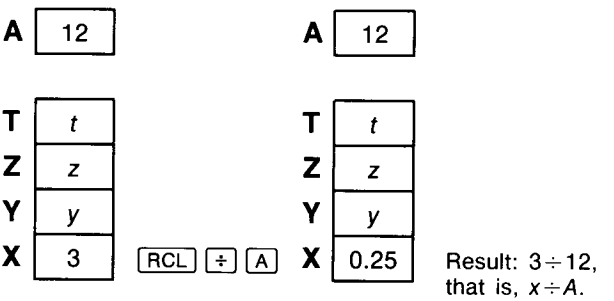


Recall Arithmetic

Recall arithmetic uses RCL +, RCL −, RCL ×, or RCL ÷ to do arithmetic in the X-register using a recalled number and to leave the result in the display. Only the X-register is affected; all other stack registers are unaffected.

$$\text{New } x = \text{Previous } x \{+, -, \times, \div\} \text{ Variable}$$

For example, suppose you want to divide the number in the X-register (3, displayed) by the value in *A* (12). Press RCL ÷ A. Now $x = 0.25$, while 12 is still in *A*.*



* Recall arithmetic saves memory space in programs. Using RCL + A (one instruction) uses half as much memory as RCL A, + (two instructions).

More Examples. Suppose the variables *D*, *E*, and *F* contain the values 1, 2, and 3. Use storage arithmetic to add 1 to each of these variables.

Keys:	Display:	Description:
1 [STO] D 2 [STO] E 3 [STO] F	3.0000	Stores the assumed values into the variables.
1 [STO] [+] D [STO] [+] E [STO] [+] F	1.0000	Adds 1 to *D*, *E*, and *F*.
■ [VIEW] D	D=2.0000	Displays the current value of *D*.
■ [VIEW] E	E=3.0000	
■ [VIEW] F	F=4.0000	
[◆]	1.0000	Clears the VIEW display; displays X-register again.

Suppose the variables *D*, *E*, and *F* contain the values 2, 3, and 4 from the last example. Divide 3 by *D*, multiply it by *E*, and add *F* to the result.

Keys:	Display:	Description:
3 [RCL] [÷] D	1.5000	Calculates $3 \div D$.
[RCL] [×] E	4.5000	$3 \div D \times E$.
[RCL] [+] F	8.5000	$3 \div D \times E + F$.

The Variable "*i*"

There is a 27th variable—the variable *i*. (The [i] is located with the [.] key.) Although it stores numbers as other variables do, it is special in that it can be used (via the [(i)] function) to refer to *other* variables—a technique called *indirect addressing*. Because this is a programming technique, it is covered in chapter 6 under "Indirectly Addressing Variables and Labels."

4

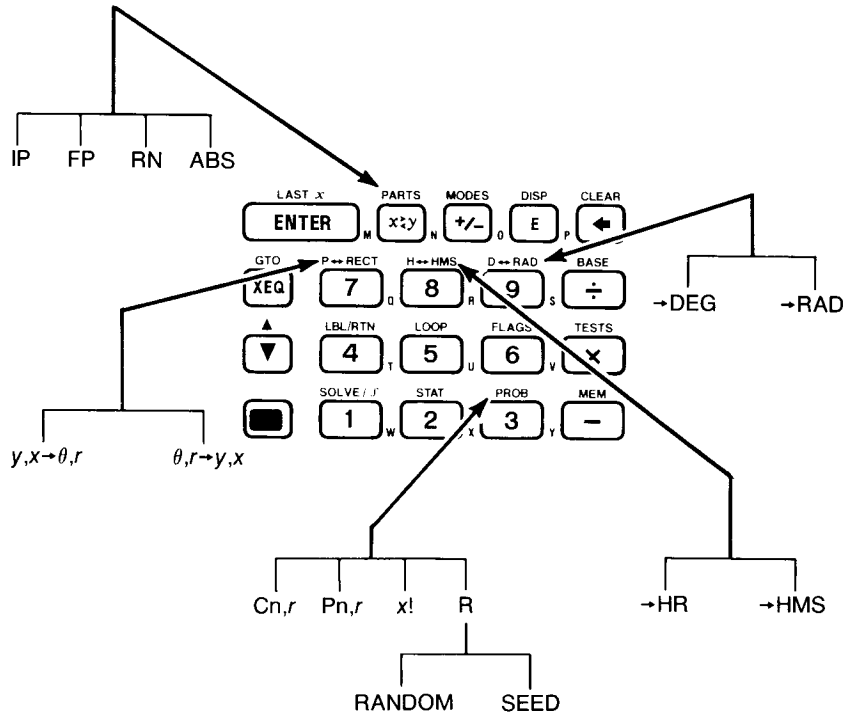
Real-Number Functions

This chapter covers most of the calculator's functions that do computations on real numbers, including some numeric functions intended for programs (such as the absolute-value function):

- Exponential and logarithmic functions.
- Trigonometric functions.
- Hyperbolic functions.
- Percentage functions.
- Conversion functions for coordinates, angles, and fractions.
- Probability functions.
- Parts of numbers (number-altering functions).

Arithmetic functions and calculations were covered in chapters 1 and 2. The advanced numeric operations (root-finding, integrating, complex numbers, base conversions, and statistics) are in part 3 of this manual.

Many of the numeric functions appear on keys in the top two rows of the keyboard. The rest appear in one of these menus:



Exponential and Logarithmic Functions

Put the number in the display first, then execute the function. There is no need to press [ENTER].

To Calculate:	Press:
Natural logarithm (base *e*)	[LN]
Common logarithm (base 10)	[■] [LOG]
Natural exponential	[e^x]
Common exponential (antilogarithm)	[■] [10^x]

The Power Function (y^x)

To calculate a number, y, raised to a power, x, key in y [ENTER] x [y^x].

For $y > 0$, x can be any rational number. For $y < 0$, x must be an integer. For $y = 0$, x must be positive.

For example:

To Calculate:	Press:	Result:
15^2	15 ■ [x^2]	225.0000
$2^{-1.4}$	2 [ENTER] 1.4 [+/-] [y^x]	0.3789
$(-1.4)^3$	1.4 [+/-] [ENTER] 3 [y^x]	−2.7440
$\sqrt[3]{2}$ or $2^{1/3}$	2 [ENTER] 3 [1/x] [y^x]	1.2599

Trigonometry

Entering π

Press ■ [π] to place the first 12 digits of π into the X-register. (The number displayed depends on the display format.) Because this is a function, π does not need to be separated from another number by [ENTER].

Note that a calculator cannot exactly represent π, since π is an irrational number.

Setting the Angular Mode

The angular mode specifies which unit of measure to assume for angles used in trigonometric functions. The mode does *not* convert numbers already present (see "Conversion Functions" in this chapter).

$$360 \text{ degrees} = 2\pi \text{ radians} = 400 \text{ grads}$$

To set an angular mode, press ■ MODES. Then select an option.

Option	Description	Annunciator
{DG}	Sets Degrees mode (DEG). Uses decimal fractions, not minutes and seconds.	none
{RD}	Sets Radians mode (RAD).	**RAD**
{GR}	Sets Grads mode (GRAD).	**GRAD**

Trigonometric Functions

With x in the display:

To Calculate:	Press:
Sine of x.	SIN
Cosine of x.	COS
Tangent of x.	TAN
Arc sine of x.	■ ASIN
Arc cosine of x.	■ ACOS
Arc tangent of x.	■ ATAN

Note

Calculations with the irrational number π cannot be expressed *exactly* with the 12-digit internal precision of the calculator. This is particularly noticeable in trigonometry; for example, the calculated sin π is not zero but -2.0676×10^{-13}, a very small number close to zero.

Example. Show that the cosine of $(5/7)\pi$ radians and the cosine of 128.57° are the same.*

Keys:	Display:	Description:
■ MODES {RD}		Sets Radians mode; RAD annunciator on.
5 ENTER 7 ÷ ■ π × COS	-0.6235	Cos $(5/7)\pi$.
■ MODES {DG} 128.57 COS	-0.6235	Switches to Degrees mode (no annunciator) and calculates cos 128.57°, which is the same as cos $(5/7)\pi$.

Programming Note. Equations using inverse trigonometric functions to determine an angle, θ, often look something like this:

$$\theta = \arctan (y/x).$$

If $x = 0$, then $y \div x$ is undefined, resulting in the error DIVIDE BY 0. For a program, then, it would be more reliable to determine θ by a *rectangular to polar conversion*, which converts x, y to r, θ. See "Coordinate Conversions," later in this chapter.

* Actually, these calculated results are the same only to four significant digits due to the inexact representation of π. (Press ■ SHOW to see more digits.)

Hyperbolic Functions

With x in the display:

To Calculate:	Press:
Hyperbolic sine of x (SINH).	[f] [HYP] [SIN]
Hyperbolic cosine of x (COSH).	[f] [HYP] [COS]
Hyperbolic tangent of x (TANH).	[f] [HYP] [TAN]
Hyperbolic arc sine of x (ASINH).	[f] [HYP] [f] [ASIN]
Hyperbolic arc cosine of x (ACOSH).	[f] [HYP] [f] [ACOS]
Hyperbolic arc tangent of x (ATANH).	[f] [HYP] [f] [ATAN]

Percentage Functions (%, %CHG)

The percentage functions are special (compared with [×] and [÷]) because they preserve the value of the base number (in the Y-register) when they return the result of the percentage calculation (in the X-register). You can then carry out subsequent calculations using both the base number and the result without reentering the base number.

To Calculate:	Key In:
x% of y	y [ENTER] x [f] [%]
Percentage change from y to x. ($y \neq 0$)	y [ENTER] x [f] [%CHG]

Example. Find the sales tax at 6% and the total cost of a $15.76 item. Use FIX 2 display format so the costs are rounded appropriately.

Keys:	Display:	Description:
[f] [DISP] {FX} 2		Rounds display to two decimal places.

15.76 [ENTER] 6 [■] [%]	0.95	Calculates 6% tax.
[+]	16.71	Total cost (base price + tax).

Suppose that the $15.76 item cost $16.12 last year. What is the percentage change from last year's price to this year's?

Keys:	**Display:**	**Description:**
16.12 [ENTER] 15.76 [■] [%CHG]	-2.23	This year's price dropped about 2.2% from last year's price.
[■] [DISP] {FX} 4	-2.2333	Restores FIX 4 format.

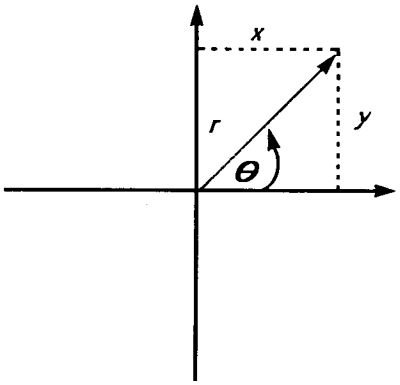
Note that the order of the two numbers is important for the %CHG function. The order affects whether the percentage change is considered positive or negative.

Conversion Functions

There are three types of conversions: coordinate (polar/rectangular), angular (degrees/radians), and fractional (decimal/minutes-seconds).

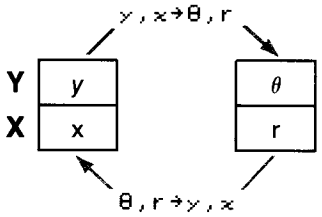
Coordinate Conversions (P↔RECT)

Rectangular coordinates (x, y) and polar coordinates (r, θ) are measured as shown in the illustration. Functions in the P↔RECT (*polar from/to rectangular*) menu convert between the two. The angle θ uses the units set by the current angular mode. A calculated result for θ will be between $-180°$ and $180°$, between $-\pi$ and π radians, or between -200 and 200 grads.

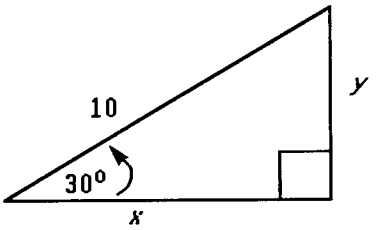


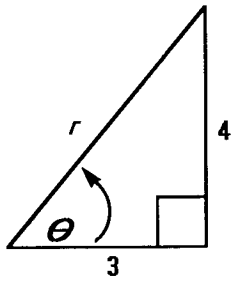
To convert between rectangular coordinates (x, y) and polar coordinates (r, θ):

1. First enter the coordinates (in rectangular form or polar form) that you want to convert. The order is y [ENTER] x or θ [ENTER] r.
2. Press [■] [P↔RECT].
3. Execute the conversion you want: {y,x→θ,r} (*rectangular to polar*) or {θ,r→y,x} (*polar to rectangular*). The converted coordinates occupy the X- and Y-registers.
4. The resulting display shows either r (polar result) or x (rectangular result). Press [x≷y] to see θ or y.



Example: Polar to Rectangular Conversion. Find x and y in the right triangle on the left. Find r and θ in the right triangle on the right.

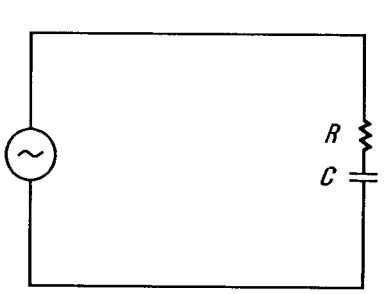


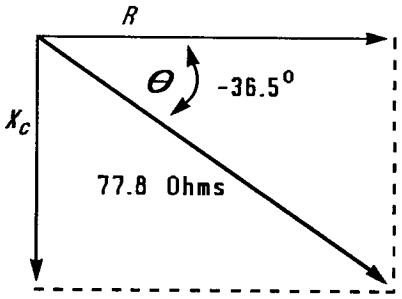


Keys:	Display:	Description:
■ MODES {DG}		Sets Degrees mode.
30 ENTER 10 ■ P↔RECT {θ,r→y,x}	8.6603	Calculates x.
x≷y	5.0000	Displays y.
4 ENTER 3 ■ P↔RECT {y,x→θ,r}	5.0000	Calculates the hypotenuse (r).
x≷y	53.1301	Displays θ.

Example: Conversion With Vectors. Engineer P.C. Bord has determined that in the RC circuit shown on the next page at left, the total impedance is 77.8 ohms and voltage lags current by 36.5°. What are the values of resistance, R, and capacitive reactance, X_C, in the circuit?

Use a vector diagram as shown with impedance equal to the polar magnitude, r, and voltage lag equal to the angle, θ, in degrees. When the values are converted to rectangular coordinates, the x-value yields R, in ohms, while the y-value yields X_C, in ohms.





Keys:	Display:	Description:
■ MODES {DG}		Sets Degrees mode.
36.5 +/- ENTER	-36.5000	Enters θ, degrees of voltage lag.
77.8	77.8_	Enters r, ohms of total impedance.
■ P↔RECT {θ,r→y,x}	62.5401	Calculates x, ohms resistance, R.
x≷y	-46.2772	Displays y, ohms reactance, X_C.

For more sophisticated operations with vectors (addition, subtraction, cross product, and dot product), refer to the "Vector Operations" program in chapter 12 ("Mathematics Programs").

Fractional Conversions (H↔HMS)

Values for time (in hours, *H*) or angles (in degrees, *D*) can be converted between a decimal-fraction form (*H.h* or *D.d*) and a minutes-seconds form (*H.MMSSss* or *D.MMSSss*) using the H↔HMS menu (*hours from/to hours-minutes-seconds*).

To convert between decimal fractions and minutes-seconds:

1. Key in the time or angle (in decimal form or minutes-seconds form) that you want to convert.
2. Press ■ [H↔HMS].
3. Select {→HR} (*hours-minutes-seconds to hours*) or {→HMS} (*hours to hours-minutes-seconds*). The result is displayed.

Example: Converting Time Formats. How many minutes and seconds are there in 1/7 of an hour? Use FIX 6 display format.

Keys:	Display:	Description:
■ [DISP] {FX} 6		
7 [1/x]	0.142857	One-seventh as a decimal fraction.
■ [H↔HMS] {→HMS}	0.083429	Equals 8 minutes and 34.29 seconds.
■ [DISP] {FX} 4		Restores FIX 4 format.

Angle Conversions (D↔RAD)

The D↔RAD (*degrees from/to radians*) menu operates independently of the angular mode. When converting to radians, the number in the X-register is assumed to be degrees. Likewise, when converting to degrees, the number in the X-register is assumed to be radians.

To convert an angle between degrees and radians:

1. Key in the angle (in decimal degrees or radians) that you want to convert.
2. Press ■ [D↔RAD].
3. Select {→DEG} (*radians to degrees*) or {→RAD} (*degrees to radians*). The result is displayed.

Probability Functions

The PROB (*probability*) menu has functions to calculate factorials, combinations, and permutations, and to obtain random numbers.

The PROB Menu

Menu Label	Description
{Cn,r}	*Combinations.* Enter *n* first, then *r.* (Nonnegative integers only.) Calculates the number of possible *sets* of *n* items taken *r* at a time. No item occurs more than once in a set, and different orders of the same *r* items are *not* counted separately.
{Pn,r}	*Permutations.* Enter *n* first, then *r.* (Nonnegative integers only.) Calculates the number of possible *arrangements* of *n* items taken *r* at a time. No item occurs more than once in an arrangement, and different orders of the same *r* items *are* counted separately.
{x!}	*Factorial* and *Gamma.* Calculates the factorial of the displayed positive integer ($0 \leqslant x \leqslant 253$).
	To calculate the gamma function of *a*, $\Gamma(a)$, key in $(a - 1)$ and press [f] [PROB] {x!}. (The {x!} function calculates $\Gamma(x + 1)$. The value for *x* cannot be a negative integer.)
{R}	*Random number generator.* Has two options. Pressing {RANDOM} generates a random number in the range $0 \leqslant x < 1$.* Pressing {SEED} starts a new random-number sequence with the number that is in the X-register.

* The random number generator in the HP-32S actually returns a number that is part of a uniformly distributed pseudo-random number sequence. It passes the spectral test (D. Knuth, *Seminumerical Algorithms*, vol. 2. London: Addison Wesley, 1981).

{RANDOM} uses a seed to generate a random number. Each random number generated becomes the seed for the next random number. Therefore, a sequence of random numbers can be repeated by starting with the same seed. You can store a new seed with the {SEED} function. If memory is cleared, the seed is reset to zero.

Example: Combinations of People. A company employing 14 women and 10 men is forming a 6-person safety committee. How many different combinations of people are possible?

Keys:	Display:	Description:
24 [ENTER] 6	6_	Twenty-four people grouped six at a time.
[PROB]	Cn,r Pn,r x! R	Probability menu.
{Cn,r}	134,596.0000	Total number of combinations possible.

If employees are chosen at random, what is the probability that the committee will contain six women? To find the *probability* of an event, divide the number of combinations *for that event* by the *total* number of combinations.

Keys:	Display:	Description:
14 [ENTER] 6	6_	Fourteen women grouped six at a time.
[PROB] {Cn,r}	3,003.0000	Number of combinations of six women on the committee.
[x≷y]	134,596.0000	Brings total number of combinations back into the X-register.
[÷]	0.0223	Divides combinations of women by total combinations to find probability that any one combination would have all women.

Parts of Numbers

The functions in the PARTS menu alter the number in the X-register in simple ways. These functions are used in programming.

The PARTS Menu

Menu Label	Description
{IP}	*Integer part.* Removes the fractional part of *x* and replaces it with zeros. (For example, the integer part of 14.2300 is 14.0000.)
{FP}	*Fractional part.* Removes the integer part of *x* and replaces it with zeros. (For example, the fractional part of 14.2300 is 0.2300.)
{RN}	*Round* (RND). Rounds *x internally* to the number of digits specified by the display format. (If not rounded, the internal number is represented by 12 digits.)
{ABS}	*Absolute value.* Replaces *x* with its absolute value.

Names of Functions

You might have noticed that the name of a function appears in the display when you press and hold the key to execute it. (The name remains displayed for as long as you hold the key down.) For instance, while pressing [√x], the display shows SQRT. "SQRT" is the name of the function as it will appear in program lines, and is the name by which the function is alphabetized in the function index.

Part 2

Programming

5

Simple Programming

Part 1 of this manual introduced you to functions and operations that you can use *manually*, that is, by pressing a key for each individual operation. A *program* lets you repeat operations or calculations without repeating the keystrokes. In this chapter you will learn how to program a series of operations to occur automatically. In the next chapter, "Programming Techniques," you will learn about subroutines and conditional instructions.

Introduction: A Simple Programming Example. To find the area of a circle with a radius of 5, you would use the formula $A = \pi r^2$ and press

5 [x²] [π] [×]

to get the result for this circle, 78.5398.

But what if you wanted to find the area of many different circles? Rather than repeat the given keystrokes each time (varying only the "5" for the different radii), you can put the repeatable keystrokes into a program:

```
001 x²
002 π
003 ×
```

This very simple program assumes that the value for the radius is in the X-register (the display) when the program starts to run. It computes the area and leaves it in the X-register.

To enter this program into program memory, do the following:

Keys:	Display:	Description
[f] [PRGM]		
[f] [GTO] [·] [·]	PRGM TOP	This resets the program pointer.
[f] [x^2] [f] [π] [×]	001 x^2 002 π 003 ×	
[f] [PRGM]		

Now try running this program to find the area of a circle with a radius of 5.

Keys:	Display:	Description:
[f] [GTO] [·] [·]		This sets the program to its beginning.
5 [R/S]	78.5398	The answer!

Creating a Program

We will continue using the above program for the area of a circle to illustrate programming concepts and methods.

Program Boundaries (LBL and RTN)

If you want more than one program stored in program memory, then a program needs a *label* to mark its beginning (such as `A01 LBL A`) and a *return* to mark its end (such as `A05 RTN`). Notice that the line numbers acquire an `A` to match their label.

Program Labels. Programs and segments of programs (called *routines*) should start with a label. To record a label, press:

[f] [LBL/RTN] {LBL} *letter-key*

The label is used as identification for executing a specific program or routine. The label is a single letter from A through Z. The letter keys are used as they are for variables (as discussed in chapter 3). You cannot assign the same label more than once (this causes the message DUPLICAT. LBL), but a label can use the same letter that a variable uses.

It is possible to have one program (the top one) in memory without any label. However, adjacent programs need a label between them to keep them distinct.

Program Line Numbers. Line numbers are preceded by the letter for the label, such as A01. If one label's routine has more than 99 lines, then the line number appears with a decimal point instead of the leftmost number, such as A.01 for line 101 in A. For more than 199 lines, the line number uses a comma, such as A,01 for line 201.

Program Returns. Programs and subroutines should end with a return instruction. The keystrokes are:

■ [LBL/RTN] {RTN}

When a program finishes running, the last RTN instruction returns the program pointer to PRGM TOP, the top of program memory.

Program Entry (PRGM)

Pressing ■ [PRGM] toggles the calculator into and out of program entry (**PRGM** annunciator on). Keystrokes during program entry are stored as program lines in memory. Each instruction or number occupies one program line, and there is no limit (other than available memory) on the number of lines in a program.

To enter a program into memory:

1. Press ■ [PRGM] for program entry.

2. Press ■ [GTO] [·] [·] to display PRGM TOP. This sets the *program pointer* to a known spot, before any other programs. As you enter program lines, they are inserted *before* all other program lines.

   If you don't need any other programs that might be in memory, clear program memory by pressing ■ [CLEAR] {PGM}. To confirm that you want *all* programs deleted, press {Y} after the message CL PGMS? Y N.

3. Give the program a *label*—a single letter, A through Z. Press ■ [LBL/RTN] {LBL} *letter*. Choose a letter that will remind you of the program, such as "A" for "area."

4. To record calculator operations as program instructions, press the same keys you would to do an operation manually. Remember that many functions don't appear on the keyboard but must be accessed using menus.

5. End the program with a *return* instruction, which sets the program pointer back to PRGM TOP after the program runs. Press ■ [LBL/RTN] {RTN}.

6. Press [C] (or ■ [PRGM]) to cancel program entry.

Numbers in program lines are displayed as precisely as you entered them, using ALL or SCI format. (If digits are hidden in a long number by the line number or an exponent, press ■ [SHOW] to view them.)

Data Input and Output. For programs that need more than one input or return more than one output, there are program instructions that will prompt for a specific variable (INPUT) and display a labeled variable (VIEW). These are covered later in this chapter under "Data Input and Output."

[C], [←], and ■ [CLEAR] {×} in Program Entry. Note these special conditions during program entry:

- [C] always cancels program entry. It never clears a number to zero.
- [←] deletes the current program line. It backspaces if a digit is being entered (cursor present).
- To *program* a function to clear the X-register, use ■ [CLEAR] {×}.

Function Names in Programs. The name of a function that is used in a program line is not necessarily the same as the function's name on its key or in its menu. The name that is used in a program is usually a fuller abbreviation than that which can fit on a key or in a menu. This fuller name appears briefly in the display whenever you execute a function—as long as you hold down the key, the name is displayed.

Example: Entering a Labeled Program. The following keystrokes delete the previous program for the area of a circle and enter a new one that includes a label and a return instruction. If you make a mistake during entry, press [◆] to delete the current program line, then reenter it correctly.

Keys:	Display:	Description:
[PRGM]		Activates program entry (**PRGM** on).
[CLEAR] {PGM} {Y}	PRGM TOP	Clears all of program memory.
[LBL/RTN] {LBL} A	A01 LBL A	Labels this program routine A (for *a*rea).*
[x^2]	A02 x^2	Enters the three program lines.
[π]	A03 π	
[×]	A04 ×	
[LBL/RTN] {RTN}	A05 RTN	Ends the program.
[C]		Cancels program entry (**PRGM** annunciator off).

* If this causes the message DUPLICAT. LBL, then use a different letter or clear the existing program A.

Running a Program

To run or *execute* a program, program entry cannot be active (no program-line numbers displayed; **PRGM** off). Pressing [C] will cancel program entry.

Executing a Program (XEQ)

Press [XEQ] *label* to execute the program labeled with that letter.* The **PRGM** annunciator blinks on and off while the program is running.

If necessary, enter the data before executing the program.

Example. Run the program labeled A to find the areas of three different circles with radii of 5, 2.5, and 2π. Remember to enter the radius before executing A.

Keys:	**Display:**	**Description:**
5 [XEQ] A	78.5398	Enters the radius, then starts program A. The resulting area is displayed.
2.5 [XEQ] A	19.6350	Calculates area of second circle.
2 ■ [π] [×] [XEQ] A	124.0251	Calculates area of third circle.

* If there is only one program in memory, you can also execute it by pressing ■ [GTO] [·] [·] [R/S] (*run/stop*).

Testing a Program

If you know there is an error in a program, but are not sure where the error is, then a good way to test the program is by stepwise execution. It is also a good idea to test a long or complicated program before relying on it. By stepping through its execution, one line at a time, you can see the result after each program line is executed, so you can verify the progress of known data whose correct results are also known.

1. As for regular execution, make sure program entry is not active (**PRGM** annunciator off).
2. Press ■ GTO *label* to set the program pointer to the start of the program (that is, at its LBL instruction). The *go to* instruction moves the program pointer without starting execution. (If the program is the first or only program, you can press ■ GTO · · to move to its beginning.)
3. Press and hold ▼. This displays the current program line. When you release ▼, the line is executed. The result of that execution is then displayed (it is in the X-register).

   To move to the *preceding* line, you can press ■ ▲. No execution occurs.
4. The program pointer moves to the next line. Repeat step 3 until you find an error (an incorrect result occurs) or reach the end of the program.

If program entry is active, then ▼ (or ■ ▲) simply changes the program pointer, without executing lines. Holding down an arrow key during program entry makes the lines roll by automatically.

Example: Testing a Program. Step through the execution of the program labeled A. Use a radius of 5 for the test data. Check that program entry is not active before you start.

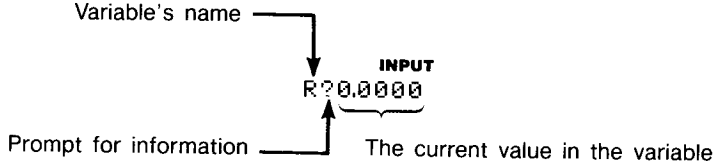
Keys:	Display:	Description:
5	5_	
■ GTO A	5.0000	Moves program counter to label A.
▼ *(hold)* *(release)*	A01 LBL A 5.0000	
▼ *(hold)* *(release)*	A02 x^2 25.0000	Squares input.
▼ *(hold)* *(release)*	A03 π 3.1416	Value of π.
▼ *(hold)* *(release)*	A04 × 78.5398	25π.
▼ *(hold)* *(release)*	A05 RTN 78.5398	End of program. Result is correct.

Data Input and Output

The calculator's *variables* are used to store data input, intermediate results, and final results. (Variables, as explained in chapter 3, are identified by a letter from *A* through *Z*, but *the variables' names have nothing to do with program labels.*)

Entering Data Into Variables (INPUT)

The INPUT instruction (■ INPUT *variable*) stops a running program and displays a prompt for the given variable. This display includes the existing value for the variable, such as



Press [R/S] (*run/stop*) or [▼] to resume the program. The value you keyed in then writes over the contents of the X-register *and* is stored in the given variable. If you have not changed the displayed value, then that value is retained in the X-register.

The area-of-a-circle program with an INPUT instruction looks like this:

```
A01 LBL A
A02 INPUT R
A03 x²
A04 π
A05 ×
A06 RTN
```

Using INPUT in a Program.

1. Decide which data values you will need, and assign them names. (In the area-of-a-circle example, the only input needed is the radius, which we can assign to *R*.)
2. In the beginning of the program, insert an INPUT instruction for each variable whose value you will need. Later in the program, when you write the part of the calculation that needs a given value, insert a [RCL] *variable* instruction to bring that value back into the stack.*

For Example: See the "Time Value of Money" program on page 222 in part 4. The first thing that routine T does is collect all the necessary input for the variables *N, I, B, P,* and *F* (lines T02 through T06).

* Since the INPUT instruction also leaves the value you just entered in the X-register, you don't *have* to recall the variable at a later time—you could INPUT it and use it when you need it. You might be able to save some memory space this way. However, in a long program it is simpler to just input all your data up front, and then recall individual variables as you need them.

Remember also that the user of the program can do calculations while the program is stopped, waiting for input. This can alter the contents of the stack, which might affect the next calculation to be done by the program. Thus the program should not assume that the X-, Y-, and Z-registers' contents will be the same before and after the INPUT instruction. If you collect all the data in the beginning and then recall them when needed for calculation, then this prevents the stack's contents from being altered just before a calculation.

When the Program Runs. When you run the program, it will stop at each INPUT, turn on the **INPUT** annunciator, and prompt you for that variable, such as R?0.0000. The value displayed (and in the X-register) will be the current contents of *R*.

- **To change the number,** key in the new number and press [R/S].* If you need to calculate a number, you can do so before pressing [R/S].
- **To leave the number unchanged,** just press [R/S].
- **To calculate with the displayed number,** press [ENTER] before keying in another number..
- **To cancel the INPUT prompt,** press [C].† The current value for the variable remains in the X-register. If you press [R/S] to resume the program, the canceled INPUT prompt is repeated.
- **To display digits hidden by the prompt,** press [■] [SHOW]. (If it is a binary number with more than 12 digits, use the [√x] and [Σ+] keys to see the rest.)

Displaying Data in Variables (VIEW)

The programmed VIEW instruction ([■] [VIEW] *variable*) stops a running program and displays and identifies the contents of the given variable, such as

A=78.5398

This is a *display only,* and it does not copy the number to the X-register.

- Pressing [ENTER] copies this number to the X-register.
- If the number has more than 10 digits, pressing [■] [SHOW] displays the entire number. (If it is a binary number with more than 12 digits, use the [√x] and [Σ+] keys to see the rest.)
- Pressing [C] (or [←]) erases the VIEW display and shows the X-register.

* This new number writes over the old value in the X-register.

† If you press [C] during digit entry, it clears the number to zero. Press it again to cancel the INPUT prompt.

■ Pressing ■ [CLEAR] clears the contents of the displayed variable.

Press [R/S] to continue the program.

For Example: See the program for "Solutions of Simultaneous Equations—Determinant Method," on page 175 in part 4. Lines S24 through S29 at the end of the S routine display the results for *X*, *Y*, and *Z*. Note also that each VIEW instruction in this program—as in all the applications programs—is preceded by a RCL instruction. The RCL instruction is not necessary, but it is convenient because it brings the VIEWed variable to the *X*-register, making it available for manual calculations. (Pressing [ENTER] while viewing a VIEW display would have the same effect.)

Example: INPUTting and VIEWing Variables in a Program. Write an equation to find the surface area and volume of a cylinder given its radius and height. Label the program C (for *cylinder*), and use the variables *S* (surface area), *V* (volume), *R* (radius), and *H* (height). Use these formulas:

$$V = \pi R^2 H$$
$$S = 2\pi R^2 + 2\pi RH = 2\,(\pi R^2 + \pi RH).$$

Keys:	Display:	Description:
■ [PRGM] ■ [GTO] [·] [·]	PRGM TOP	Program entry; sets pointer to top of memory.
■ [LBL/RTN] {LBL} C (C is the [LN] key)	C01 LBL C	Labels program.
■ [INPUT] R	C02 INPUT R	Instructions to prompt for radius and height, to calculate the volume, and to store volume in *V*.
■ [INPUT] H	C03 INPUT H	
[RCL] R	C04 RCL R	
■ [x^2]	C05 $\times^2$	
[RCL] [×] H	C06 RCL× H	
■ [π]	C07 π	
[×]	C08 ×	
[STO] V	C09 STO V	
[RCL] [÷] H	C10 RCL÷ H	Converts $\pi R^2 H$ to πR^2.

[RCL] R	C11 RCL R	
[RCL] [×] H	C12 RCL× H	
[■] [π]	C13 π	
[×]	C14 ×	Calculates πRH.
[+]	C15 +	Calculates ($\pi R^2 + \pi RH$).
2	C16 2_	Calculates 2 ($\pi R^2 + \pi RH$) and stores resulting surface area in *S*.
[×]	C17 ×	
[STO] S	C18 STO S	
[■] [VIEW] V	C19 VIEW V	Will display volume and surface area.
[■] [VIEW] S	C20 VIEW S	
[■] [LBL/RTN] {RTN}	C21 RTN	Ends program.
[C]		Cancels program entry.
[■] [MEM] {PGM}	LBL C 031.5	Checks memory usage and checksum. A different checksum means the program was not entered exactly as it is given here.
[■] [SHOW] *(hold)*	CHKSUM=4602	

Now find the volume and surface area of a cylinder with a radius of 2.5 cm and a height of 8.0 cm.

Keys:	**Display:**	**Description:**
[XEQ] C (C is the [LN] key)	R?0.0000	Starts executing C; prompts for *R*. (It displays whatever value happens to be in *R*.)
2.5 [R/S]	H?0.0000	Prompts for *H*. (It displays whatever value happens to be in *H*.)
8 [R/S]	V=157.0796	Resulting volume in cm^3.
[R/S]	S=164.9336	Resulting surface area in cm^2.

Stopping or Interrupting a Program

Programming a Stop or Pause (STOP, PSE)

- Pressing [R/S] (*run/stop*) during program entry inserts a STOP instruction. This will halt a running program until you resume it by pressing [R/S] from the keyboard. You can use STOP rather than RTN in order to end a program without returning the program pointer to the top of memory.
- Pressing [■] [LBL/RTN] {PSE} during program entry inserts a PSE (*pause*) instruction. This will suspend a running program for about 1 second and display the contents of the X-register.

Interrupting a Running Program

You can interrupt a running program at any time by pressing [C] or [R/S]. The program completes its current instruction before stopping. Press [R/S] (*run/stop*) to resume the program.

If you interrupt a program and then press [XEQ], [GTO], or {RTN}, you *cannot* resume the program with [R/S]. Re-execute the program instead ([XEQ] *label*).

Error Stops

If an error occurs in the course of a running program, program execution halts and an error message appears in the display. (There is a list of messages and conditions in front of the indexes.)

To see the line in the program containing the error-causing instruction, press [■] [PRGM]. The program will have stopped at that point. (For instance, it might be a ÷ instruction, which caused an illegal division by zero.)

Editing a Program

You can modify a program in program memory by inserting and deleting program lines. Even if a program line requires only a minor change, you must delete the old line and insert a new one.

To delete a program line:

1. Select the relevant program or routine (■ GTO *label*), activate program entry (■ PRGM), and press ▼ or ■ ▲ to locate the program line that must be changed. Hold the arrow key down to continue scrolling. (If you know the line number you want, pressing ■ GTO . *label nn* moves the program pointer there.)
2. Delete the line you want to change by pressing ◄. The pointer then moves to the *preceding* line. (If you are deleting more than one consecutive program line, start with the *last* line in the group.)
3. Key in the new instruction, if any. This replaces the one you deleted.
4. Exit program entry (C or ■ PRGM).

To insert a program line: Locate and display the program line that is *before* the spot where you would like to insert a line. Key in the new instruction; it is inserted *after* the currently displayed line.

For example, if you wanted to insert a new line between lines A04 and A05 of a program, you would first display line A04, then key in the instruction or instructions. Subsequent program lines, starting with the original line A05, are moved *down* and renumbered accordingly.

Program Memory

Viewing Program Memory

Pressing ■ [PRGM] toggles the calculator into and out of program entry (**PRGM** annunciator on, program lines displayed). When program entry is active, the contents of program memory are displayed.

Program memory starts at PRGM TOP. The list of program lines is circular, so you can wrap the program pointer from the bottom to the top and reverse. While program entry is active, there are three ways to change the program pointer (the displayed line):

- Use the arrow keys, [▼] and ■[▲]. Pressing [▼] at the last line wraps the pointer around to PRGM TOP, while pressing ■[▲] at PRGM TOP wraps the pointer around to the last program line.

  To move more than one line at a time ("scrolling"), continue to *hold* the [▼] or [▲] key.
- Press [GTO] [·][·] to move the program pointer to PRGM TOP.
- Press [GTO] [·] *label nn* to move to a labeled line number < 100.

If program entry is not active (no lines displayed), you can also move the program pointer by pressing ■[GTO] *label*.

Canceling program entry does *not* change the position of the program pointer.

Memory Usage

Each program line uses either 1.5 or 9.5 bytes.

- Numbers use 9.5 bytes, *except* for integer numbers from zero through 99, which use only 1.5 bytes.
- All other instructions use 1.5 bytes.

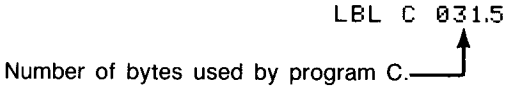
If during program entry you encounter the message MEMORY FULL, then there is not enough room in program memory for the line you just tried to enter. You can make more room available by clearing programs or other data. See "Clearing One or More Programs" below, or "Managing Calculator Memory" in appendix B.

The Catalog of Programs (MEM)

The catalog of programs is a list of all program labels with the number of bytes of memory used by each label and the lines associated with it. Press ■ [MEM] {PGM} to display the catalog, and press [▼] or ■ [▲] to move within the list. You can use this catalog to:

- Review the labels in program memory and the memory cost of each labeled program or routine.
- Execute a labeled program. (Press [XEQ] or [R/S] while the label is displayed.)
- Display a labeled program. (Press ■ [PRGM] while the label is displayed.)
- Delete specific programs. (Press ■ [CLEAR] while the label is displayed.)
- See the checksum associated with a given program segment. (Press ■ [SHOW].)

The catalog shows you how many bytes of memory each labeled program segment uses. The programs are identified by program label:



Clearing One or More Programs

To clear (delete from memory) a specific program:

1. Press ■ [MEM] {PGM} and display (using [▼] and ■ [▲]) the label of the program.

2. Press ■ [CLEAR].
3. Press [C] to cancel the catalog or [←] to back out.

To clear all programs in memory:

1. Press ■ [PRGM] to display program lines (**PRGM** annunciator on).
2. Press ■ [CLEAR] {PGM} to clear program memory.
3. The message CL PGMS? Y N prompts you for confirmation. Press {Y}.
4. Press ■ [PRGM] to cancel program entry.

Clearing all of memory (■ [CLEAR] {ALL}) also clears all programs.

The Checksum

The *checksum* is a unique hexadecimal value given to each program label and its associated lines (until the next label). This number is useful for comparison with a known checksum for an existing program that you have keyed into program memory. If the known checksum and the one shown by your calculator are the same, then you have correctly entered all the lines of the program. To see your checksum:

1. Press ■ [MEM] {PGM} for the catalog of program labels.
2. Display the appropriate label by using the arrow keys, if necessary.
3. Press and hold ■ [SHOW] to display CHKSUM=*value*.

For example, to see the checksum for the current program (the "cylinder" program):

Keys:	**Display:**	**Description:**
■ [MEM] {PGM}	LBL C 031.5	Displays label C, which takes 31.5 bytes.
■ [SHOW] *(hold)*	CHKSUM=4602	If your checksum does *not* match this number, then you have not entered this program correctly.

You will see that all of the application programs provided in part 4 include CHKSUM values with each labeled routine so that you can verify the accuracy of the program entry.

Nonprogrammable Functions

The following functions of the HP-32S are not programmable:

- [■] [CLEAR] {PGM}
- [■] [CLEAR] {ALL}
- [◀]
- [▼], [■] [▲]
- [■] [PRGM]
- [■] [GTO] [·] [·]
- [■] [GTO] [·] *label nn*
- [■] [MEM]
- [■] [SHOW]

Polynomial Expressions and Horner's Method

Some expressions, such as polynomials, use the same variable several times for their solution. For example, the expression

$$f(x) = Ax^4 + Bx^3 + Cx^2 + Dx + E$$

uses the variable x four different times. A program to solve such an equation could repeatedly recall a stored copy of x from a variable. A shorter programming method, however, would be to use a stack which has been filled with the constant (see "Filling the Stack With a Constant," on page 39 in chapter 2).

Horner's Method is a useful means of rearranging polynomial expressions to cut calculation steps and calculation time. It is especially expedient with SOLVE and ∫FN, two relatively complex operations that use subroutines.

This method involves rewriting a polynomial expression in a nested fashion to eliminate exponents greater than 1:

$$Ax^4 + Bx^3 + Cx^2 + Dx + E$$

$$(Ax^3 + Bx^2 + Cx + D)x + E$$

$$((Ax^2 + Bx + C)x + D)x + E$$

$$(((Ax + B)x + C)x + D)x + E$$

Example. Write a program for $5x^4 + 2x^3$ as $(((5x + 2)x)x)x$, then evaluate it for $x = 7$.

Keys:	**Display:**	**Description:**
[PRGM]		You can skip the [GTO] if the display already shows PRGM TOP.
[GTO] [·] [·]	PRGM TOP	
[LBL/RTN] {LBL} P	P01 LBL P	
[INPUT] X	P02 INPUT X	Fills the stack with x, then calculates $5x$.
[ENTER]	P03 ENTER	
[ENTER]	P04 ENTER	
[ENTER]	P05 ENTER	
5	P06 5	
[×]	P07 ×	
2	P08 2	
[+]	P09 +	
[×]	P10 ×	$(5x + 2)x$.
[×]	P11 ×	
[×]	P12 ×	$(5x + 2)x^3$.
[LBL/RTN] {RTN}	P13 RTN	
[C]		Cancels program entry.

Now evaluate this polynomial for $x = 7$.

XEQ P	X?*value*	Prompts for x.
7 R/S	12,691.0000	Result.

A more general form of this program for any equation $(((Ax + B)x + C)x + D)x + E$ would be:

```
P01 LBL P
P02 INPUT A
P03 INPUT B
P04 INPUT C
P05 INPUT D
P06 INPUT E
P07 INPUT X
P08 ENTER
P09 ENTER
P10 ENTER
P11 RCL× A
P12 RCL+ B
P13 ×
P14 RCL+ C
P15 ×
P16 RCL+ D
P17 ×
P18 RCL+ E
P19 RTN
```

6

Programming Techniques

Chapter 5 covered the basics of programming. This chapter delves into more sophisticated but useful techniques:

- Using subroutines to simplify programs by separating and labeling portions of the program that are dedicated to particular tasks. The use of subroutines also shortens a program that must perform a series of steps more than once.
- Using conditional instructions (comparisons and flags) to determine which instructions or subroutines should be used in a particular case.
- Using loops with counters to execute a set of instructions a certain number of times.
- Using indirect addressing to access different variables using the same program instruction.

Routines in Programs

A program is composed of one or more *routines*. A routine is a functional unit that accomplishes something specific. Complicated programs need routines to group and separate tasks. This makes a program easier to write, read, understand, and alter.

For example, look at the program for "Normal and Inverse-Normal Distributions" on page 215 in part 4. This program has four routines, labeled S, D, N, and F. Routine S "initializes" the program by collecting the input for the mean and standard deviation. Routine D sets a limit of integration, executes routine N, and displays the result. Routine N integrates the function defined in routine F and finishes the probability calculation of $Q(x)$.

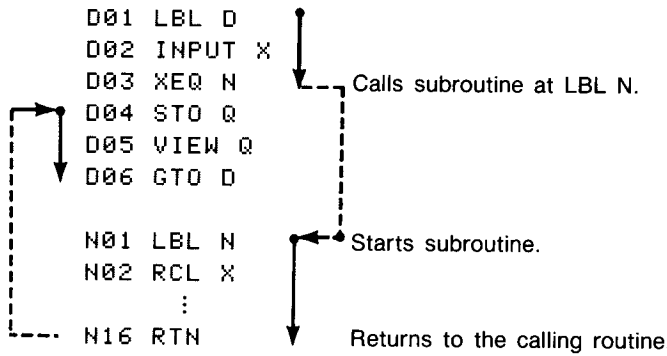
A routine typically starts with a label (LBL) and ends with an instruction that alters or stops program execution, such as RTN, GTO, or STOP, or perhaps another label.

Calling Subroutines (XEQ, RTN)

A *subroutine* is a routine that is *called from* (executed by) another routine and *returns to* that same routine when the subroutine is finished. The subroutine *must* start with a LBL and end with a RTN. A subroutine is itself a routine, and it can call other subroutines.

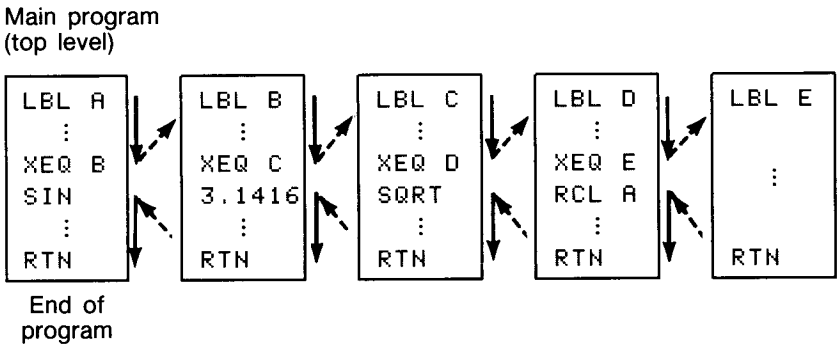
- XEQ must branch to a label (LBL) for the subroutine. (It cannot branch to a line number.)
- At the very next RTN encountered, program execution returns to the line after the originating XEQ.

For example, routine N in the "Normal and Inverse-Normal Distributions" program is a subroutine (to calculate $Q(x)$) that is called from routine D by line `D03 XEQ N`. Routine N ends with a RTN instruction that sends program execution back to routine D (to store and display the result) at line D04.



Nested Subroutines

A subroutine can call another subroutine, and that subroutine can call yet another subroutine. This "nesting" of subroutines—the calling of a subroutine within another subroutine—is limited to a stack of subroutines seven levels deep (not counting the topmost program level). The operation of nested subroutines is as shown below:

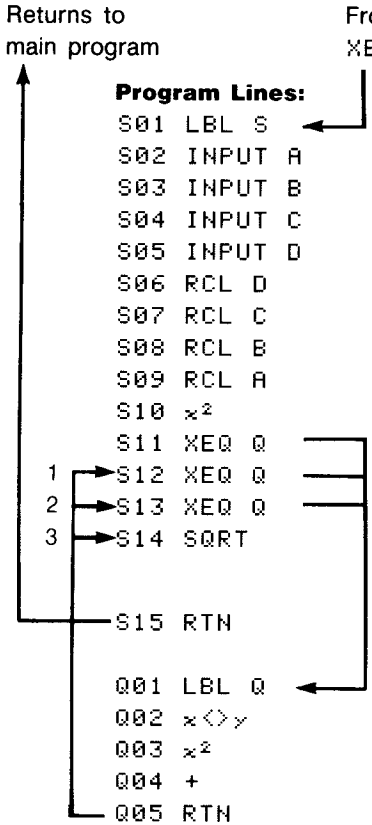


Attempting to execute a subroutine nested more than seven levels deep causes an `XEQ OVERFLOW` error.

Example: A Nested Subroutine. The following subroutine, labeled S, calculates the value of the expression

$$\sqrt{a^2 + b^2 + c^2 + d^2}$$

as part of a larger calculation in a larger program. The subroutine calls upon *another* subroutine (a nested subroutine), labeled Q, to do the repetitive squaring and addition. This keeps the program shorter than it would be without the subroutine.



Branching (GTO)

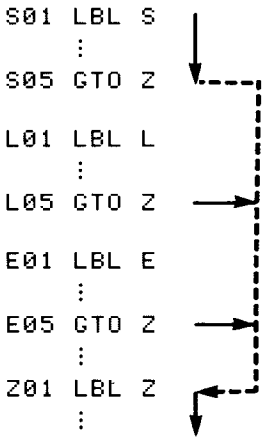
As we have seen with subroutines, it is often desirable to transfer execution to a part of the program other than the next line. This is called *branching*.

Unconditional branching uses the GTO (*go to*) instruction to branch to a program *label*. It is not possible to branch to a specific line number during a program.

[■] [GTO] *label*

A Programmed GTO Instruction. The GTO *label* instruction transfers the execution of a running program to the program line containing that label, wherever it may be. The program continues running from the new location, and it does *not* ever automatically return to its point of origination, so GTO is not used for subroutines.

For example, consider the "Curve Fitting" program on page 204 in part 4. The `GTO Z` instruction branches execution from any one of three independent initializing routines to LBL Z, the routine that is the common entry point into the heart of the program:



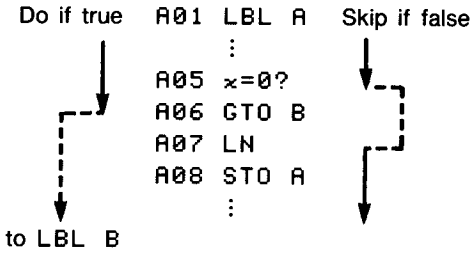
Using [■] [GTO] From the Keyboard. If program entry is *not* active (no program lines displayed; **PRGM** off), then you can use [■] [GTO] to move the program pointer to a specified label or line number *without* starting program execution.

- To a label: [■] [GTO] *label* (Example: [■] [GTO] A .)
- To a line number: [■] [GTO] [.] *label nn* (*nn* < 100. Example: [■] [GTO] [.] A05.)
- To `PRGM TOP`: [■] [GTO] [.][.]

Conditional Instructions

Another way to alter the sequence of program execution is by a *conditional test*, a true/false test that compares two numbers and skips the next program instruction if the proposition is false.

For instance, if a conditional instruction on line A05 is x=0? (that is, *is x equal to zero?*), then the program compares the contents of the X-register with zero. If the X-register *does* contain zero, then the program goes on to the next line. If the X-register does *not* contain zero, then the program *skips* the next line, thereby branching to line A07. This rule is commonly known as "Do if true."



The example above points out a common technique used with conditional tests: the line immediately after the test (which is only executed in the "true" case) is a *branch* to another label. So the net effect of the test is to branch to a different routine under certain circumstances.

There are three categories of conditional instructions:

- Comparison tests. These compare the X- and Y-registers, or the X-register and zero.
- Flag tests. These check the status of flags, which can be either set or clear.
- Loop counters. These are usually used to loop a specified number of times.

Tests of Comparison (TESTS)

There are eight comparisons available for programming in the TESTS menu. Pressing ■ [TESTS] displays the two categories of tests:

Remember that x refers to the number in the X-register, and y refers to the number in the Y-register. These do *not* compare the *variables* X and Y.

Select the category of comparison, then press the menu key for the conditional instruction you want:

The TESTS Menu Keys

{x?y}	{x?0}
{≠y} for x≠y?	{≠0} for x≠0?
{<y} for x<y?	{<0} for x<0?
{>y} for x>y?	{>0} for x>0?
{=y} for x=y?	{=0} for x=0?

Although you can display these menus outside of program entry, these functions have no purpose outside of programs.

For Example: The "Quadratic Equation" program on page 191 in part 4 uses the x=0? and x<0? conditionals in routine Q.

```
Q01 LBL Q
Q02 INPUT A
Q03 x=0?          Checks the validity of A, which cannot be
                  zero.
Q04 GTO Q         If A = 0, then the program starts over.
Q05 INPUT B       If A ≠ 0, then the program continues.
      ⋮
```

Lines Q14 through Q19 calculate $B^2 - 4AC$. The following lines test for a negative value (which would produce an imaginary root).

```
      :
Q20 x<0?          Is result negative?
Q21 GTO I         If yes, branches to different routine.
Q22 SQRT          If positive, takes square root.
      :
```

Flags

A flag is an indicator of status. It is either *set* (*true*) or *clear* (*false*). *Testing a flag* is another conditional test that follows the "Do if true" rule: program execution proceeds directly if the tested flag is set, and it skips one line if the flag is clear.

Meanings of Flags. The HP-32S has seven flags, numbered 0 through 6. All of these flags can be set, cleared, and tested by a program instruction. You can also set and clear flags from the keyboard.*

- Flags 0, 1, 2, 3, and 4 have no preassigned meanings. That is, their status will mean whatever you define it to mean in a given program. (See the example below.)
- Flag 5, when set, will interrupt a program when an overflow occurs within the program, displaying OVERFLOW and ⚠.† If flag 5 is clear, a program with an overflow is not interrupted, though OVERFLOW is displayed briefly when the program eventually stops.
- Flag 6 is *automatically* set by the calculator any time an overflow occurs (although you can also set flag 6 yourself). It has no effect, but can be tested.

* The only other action that clears flags is the three-key memory clearing operation described in appendix B.

† An *overflow* occurs when a result exceeds the largest number that the calculator can handle. The largest possible number is substituted for the overflow result.

Flags 5 and 6 allow you to control overflow conditions that occur during a program. Setting flag 5 stops a program at the line just after the line that caused the overflow. By testing flag 6 in a program, you can alter the program's flow or change a result anytime an overflow occurs.

Annunciators for Set Flags. Flags 0, 1, 2, and 3 have annunciators in the display that turn on when the corresponding flag is set. The presence or absence of **0**, **1**, **2**, or **3** lets you know at any time whether any of these four flags is set or not. However, there is no such indication for the status of flags 4, 5, and 6. These flags' status can be determined only by a programmed FS? instruction. (See "Testing Flags (FS?)" below.)

Functions for Flags. Pressing ■ [FLAGS] displays the FLAGS menu:

```
SF   CF   FS?
```

After selecting the function you want, you will be prompted for the flag number, 0–6. For example, to set flag 0, press ■ [FLAGS] {SF} 0.

The FLAGS Menu

Menu Key	Description
{SF} *n*	*Set flag.* Sets flag *n*.
{CF} *n*	*Clear flag.* Clears flag *n*.
{FS?} *n*	*Is flag set?* Tests the status of flag *n*.

Testing Flags (FS?). A flag test is a conditional test that affects program execution just as the comparison tests do. The FS? *n* instruction tests whether the given flag is set. If it is, then the next line in the program is executed. If it is not, then the next line is skipped. This is the "Do if True" rule, illustrated on page 95 under "Conditional Instructions."

Although you can execute {FS?} outside of program entry, testing flags has no purpose outside of programs.

It is good practice in a program to make sure that any conditions you will be testing start out in a known state. Current flag settings depend on how they have been left by earlier programs that have been run. You should not *assume* that any given flag is clear, for instance, and that it will be set only if something in the program sets it. You should make *sure* of this by clearing the flag before the condition arises that might set it. See the example below.

Example: Using Flags. The "Quadratic Equation" program on page 191 in part 4 uses flag 0 in conjunction with the x<0? comparison to remember the sign of *B*. Note that line Q11 clears flag 0 to make sure that it will be set for *only* the condition desired.

```
Q11 CF 0          Makes sure that flag 0 is clear.
Q12 x<0?          Is B (in X-register) negative?
Q13 SF 0          Sets flag 0 if B is negative.
     ⋮
Q23 FS? 0         Is flag 0 set (is B negative)?
Q24 +/-           If yes, change sign.
Q25 +             In either case, add.
     ⋮
```

Other programs in part 4 that make use of flags are "Curve Fitting" and "Unit Conversions." They both use flags to remember which condition the user wants solved (which type of curve, which type of conversion), thereby affecting which options or calculations are chosen.

Loops (GTO, LOOP)

Branching backwards—that is, to a label in a previous line—makes it possible to execute part of a program more than once. This is called *looping*.

```
D01 LBL D
D02 INPUT M
D03 INPUT N
D04 INPUT T
D05 GTO D
```

This routine (taken from the "Coordinate Transformations" program on page 198 in part 4) is an example of an *infinite loop*. It is used to collect the initial data prior to the coordinate transformation. After entering the three values, it is up to the user to manually interrupt this loop by selecting the transformation to be performed (pressing XEQ N for the old-to-new system or XEQ O for the new-to-old system).

Conditional Loops (GTO)

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

For example, the following routine uses a loop to diminish a value *A* by a constant amount *B* until the resulting *A* is less than or equal to *B*.

Program Lines:	**Description:**
A01 LBL A	
A02 INPUT A	
A03 INPUT B	
S01 LBL S	
S02 RCL A	It is easier to recall *A* than to remember where it is in the stack.
S03 RCL- B	Calculates $A - B$.
S04 STO A	Replaces old *A* with new result.
S05 RCL B	Recalls constant for comparison.
S06 x<y?	Is $B <$ new *A*?
S07 GTO S	Yes: loops to repeat subtraction.
S08 VIEW A	No: displays new *A*.
S09 RTN	

Loops With Counters (DSE, ISG)

When you want to execute a loop a specific number of times, use the DSE (*decrement; skip if less than or equal to*) or ISG (*increment; skip if greater than*) conditional functions in the LOOP menu (■ LOOP). Each time a LOOP function is executed in a program, it automatically *decrements* or *increments* a counter value stored in a variable. It compares the current counter value to a final counter value, then continues or exits the loop depending on the result.

For a count-down loop, use:

■ LOOP {DSE} *variable*

For a count-up loop, use:

■ LOOP {ISG} *variable*

These functions accomplish the same thing as a FOR-NEXT loop in BASIC:

```
FOR variable = initial-value TO final-value STEP increment
      ⋮
NEXT variable
```

A DSE instruction is like a FOR-NEXT loop with a negative increment.

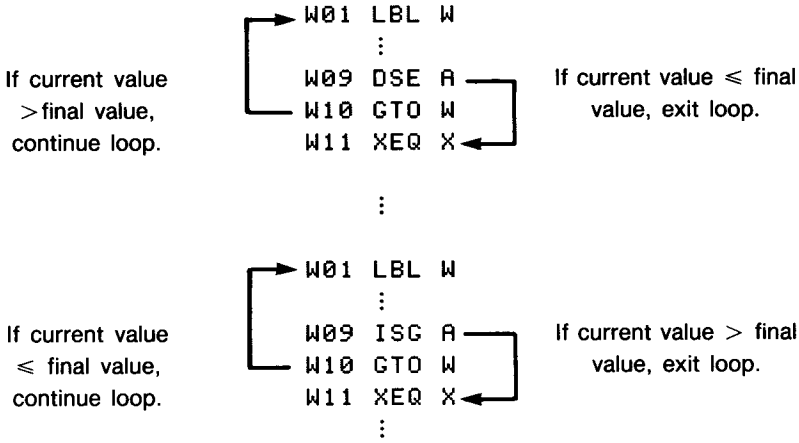
After pressing the menu key for {DSE} or {ISG}, you will be prompted for a variable that will contain the *loop-control number* (described below).

The Loop-Control Number. The specified variable should contain a loop-control number ±*ccccccc.fffii*, where:

- ±*ccccccc* is the current counter value (1 to 12 digits). This value *changes* with loop execution.
- *fff* is the final counter value (must be three digits). This value does *not* change as the loop runs.
- *ii* is the interval for incrementing and decrementing (must be two digits or unspecified). This value does *not* change. An unspecified value for *ii* is assumed to be 01 (increment/decrement by 1).

Given the loop-control number *ccccccc.fffii,* DSE decrements *ccccccc* to *ccccccc* − *ii,* compares the new *ccccccc* with *fff,* and makes program execution skip the next program line if this *ccccccc* ⩽ *fff.*

Given the loop-control number *ccccccc.fffii,* ISG increments *ccccccc* to *ccccccc* + *ii,* compares the new *ccccccc* with *fff,* and makes program execution skip the next program line if this *ccccccc* > *fff.*



For example, the loop-control number 0.050 for ISG means: start counting at zero, count up to 50, and increase the number by 1 each loop.

The following program uses ISG to loop 10 times. The loop counter (0000001.01000) is stored in the variable Z. Leading and trailing zeros can be left off.

```
L01 LBL L
L02 1.01
L03 STO Z
M01 LBL M
M02 ISG Z
M03 GTO M
M04 RTN
```

Press ■ VIEW Z to see that the loop-control number is now 11.0100.

Indirectly Addressing Variables and Labels

Indirect addressing is a technique used in advanced programming to specify a variable or label *without specifying beforehand exactly which one*. This is determined when the program runs, so it depends on the intermediate results (or input) of the program.

Indirect addressing uses two different keys: [i] (with [·]) and [(i)] (with [R/S]).* These keys are active for many functions that take *A* through Z as variables or labels.

- *i* is a variable whose contents can refer to *another* variable or label. It holds a number just like any other variable (*A* through Z).
- [(i)] is a programming function that directs, "Use the number in *i* to determine which variable or label to address." This is an *indirect address*. (A through Z are *direct addresses*.)

Both [i] and [(i)] are used together to create an indirect address. (See the examples below.) By itself, *i* is just another variable. By itself, [(i)] is either undefined (no number in *i*) or uncontrolled (using whatever number happens to be left over in *i*).

The Variable "*i*"

You can store, recall, and manipulate the contents of *i* just as you can the contents of other variables. You can even solve for *i* and integrate using *i*.

Functions That Use *i* Directly

STO *i*	INPUT *i*	DSE *i*
RCL *i*	VIEW *i*	ISG *i*
STO +,−,×,÷ *i*	∫FN *i*	
RCL +,−,×,÷ *i*	SOLVE *i*	

* The variable *I* has nothing to do with [(i)] or the variable *i*.

The Indirect Address, (*i*)

Many functions that use A through Z (as variables or labels) can use [(i)] to refer to A through Z (variables or labels) *indirectly*. The function [(i)] uses the value in variable *i* to determine which variable or label to address. This table shows how:

Indirect Addressing

If *i* contains:	Then (*i*) will address:
± 1	variable *A* or label A
⋮	⋮
±26	variable *Z* or label Z
≥27 or ≤−27 or 0	error: INVALID (i)

Only the absolute value of the integer portion of the number in *i* is used for addressing.

Following are the functions that can use (*i*) as an address. For GTO, XEQ, and FN=, (*i*) refers to a label; for all other functions it refers to a variable.

Functions That Use (*i*) for Indirect Addressing

STO(i)	INPUT(i)
RCL(i)	VIEW(i)
STO+,−,×,÷ (i)	DSE(i)
RCL+,−,×,÷ (i)	ISG(i)
XEQ(i)	SOLVE(i)
GTO(i)	∫FN(i)
	FN=(i)

Program Control With (*i*)

Since the contents of *i* can change each time a program runs—or even in different parts of the same program—a program instruction such as GTO(i) can branch to a different label at different times. This maintains flexibility by leaving open (until the program runs) exactly which variable or program label will be needed. (See the first example below.)

Indirect addressing is very useful for counting and controlling loops. The variable *i* serves as an *index*, holding the address of the variable that contains the loop-control number for the functions DSE and ISG. (See the second example below.)

Example: Choosing Subroutines With (*i*). The "Curve Fitting" program on page 204 in part 4 uses indirect addressing to determine which model to use to compute estimated values for x and y. (Different subroutines compute x and y for the different models.) Notice that *i* is stored and then indirectly addressed in widely separated parts of the program.

The first four routines (S, L, E, P) of the program specify the curve-fitting model that will be used and assign a number (1, 2, 3, 4) to each of these models. This number is then stored during routine Z, the common entry point for all models:

```
Z03 STO i
```

Routine Y uses *i* to call the appropriate subroutine (by model) to calculate the x- and y-estimates. Line Y03 calls the subroutine to compute $\hat{y}$:

```
Y03 XEQ(i)
```

and line Y08 calls a different subroutine to compute $\hat{x}$ after *i* has been increased by 6:

```
Y06 6
Y07 STO+ i
Y08 XEQ(i)
```

If *i* holds:	Then XEQ(i) calls:	To:
1	LBL A	Compute *y* for straight-line model.
2	LBL B	Compute *y* for logarithmic model.
3	LBL C	Compute *y* for exponential model.
4	LBL D	Compute *y* for power model.
7	LBL G	Compute *x* for straight-line model.
8	LBL H	Compute *x* for logarithmic model.
9	LBL I	Compute *x* for exponential model.
10	LBL J	Compute *x* for power model.

Example: Loop Control With (*i*). An index value in *i* is used by the program "Solutions of Simultaneous Equations—Determinant Method" on page 175 in part 4. This program uses the looping instructions `ISG i` and `DSE i` in conjunction with the indirect instructions `RCL(i)` and `STO(i)` to fill and manipulate a matrix.

The first part of this program is routine A, which puts the initial loop-control number in *i*.

Program Lines:	Description:
`A01 LBL A`	The starting point for data input.
`A02 1.012`	Loop-control number: loop from 1 to 12 in intervals of 1.
`A03 STO i`	Stores loop-control number in *i*.

The next routine is L, a loop to collect all 12 known values for a 3×3 coefficient matrix (variables *A–I*) and the three constants (*J–L*) for the equations.

Program Lines:	Description:
L01 LBL L	This routine collects all known values in three equations.
L02 INPUT(i)	Prompts for and stores a number into the variable addressed by *i*.
L03 ISG i	Adds 1 to *i* and repeats the loop until *i* reaches 13.012.
L04 GTO L	
L05 GTO A	When *i* exceeds the final counter value, execution branches back to A.

Part 3

Advanced Operation

7

Solving for an Unknown Variable in an Equation

The SOLVE operation can solve for any one variable in an equation. For instance, take the function

$$x^2 - 3y.$$

This function can be set equal to zero to create the equation:

$$x^2 - 3y = 0.*$$

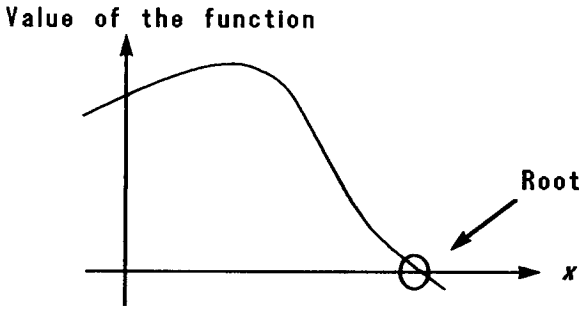
If you know the value of y in this equation, then SOLVE can solve for the unknown x. If you know the value of x, then SOLVE can solve for the unknown y. This works for "word problems" just as well:

$$Markup \times Cost = Price$$
$$Markup \times Cost - Price = 0.$$

If you know any two of these variables, then SOLVE can calculate the value of the third.

When the equation has only one variable, or when known values are supplied for all the variables except one, then to solve for x is to find the root(s) of the equation. A *root* of an equation occurs where the graph of the function crosses the x-axis, because at that point the value of the function equals zero.

* Actually, you can set the function equal to any real value, such as $x^2 - y = 10$. This can then be expressed as $x^2 - y - 10 = 0$ to use SOLVE.



Using SOLVE

To solve for an unknown variable:

1. Enter a program that defines the function. (See "Writing Programs for SOLVE," below.)
2. Select the program that defines the function to solve: press
   [f] [SOLVE/∫] {FN} *label*.
3. Solve for the unknown variable: press
   [f] [SOLVE/∫] {SOLVE} *variable*.

You can halt a running calculation by pressing [C] or [R/S].

Initial Guesses. For certain functions it helps to provide one or two *initial guesses* (in the variable and the X-register) for the unknown variable before starting the calculation (step 3). This can speed up the calculation, direct the answer toward a realistic solution, and find more than one solution, if appropriate. See "Choosing Initial Guesses for SOLVE" on page 120.

Results. The X-register and the variable itself contain the final estimate of the root, the Y-register contains the previous estimate, and the Z-register contains the value of the function at the last estimate of the root (which should be zero).

For some complicated mathematical conditions, a definitive solution cannot be found. See "Interpreting Results" and "When SOLVE Cannot Find a Root" in appendix C.

To solve for a different unknown in the same equation: Just specify the unknown variable: ■[SOLVE/∫] {SOLVE} *variable*. The same program that was last specified (FN= *label*) will be used again.

Writing Programs for SOLVE

Before you solve for an unknown variable, you must write a program or subroutine that evaluates the function.*

Writing a Function From an Equation. First simplify the equation by combining all like variables and all constants. Then move all the terms to one side of the equation, leaving only zero on the other side.

For example, the equation for the volume of a box is given by

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

Rearranging the terms to make one side equal to zero yields

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

$$L \times W \times H - V = 0.$$

To write a program evaluating a function:

1. Begin with a label so that the program can be called by SOLVE.
2. Include an INPUT instruction for each variable, including the unknown. (If there is only one variable in the function, omit the INPUT instruction since it is ignored for the unknown anyway.)†

* SOLVE works only with real numbers. However, if you have a complex-valued function that can be written to isolate the real and imaginary parts, SOLVE can then solve for the parts separately.

† The INPUT instructions are useful for multi-variable functions. Since the INPUT for the unknown is ignored, you need write only *one* program, which contains INPUT instructions for *all* variables. You can use the same program no matter which variable is the unknown.

3. Enter the instructions to evaluate the function. Use a RCL instruction any place a variable's value is needed for a calculation.
4. End the program with a RTN. The program should end with the value of the function in the X-register.

Each time that SOLVE executes your program (which could be many times), the value of the unknown variable changes, as does the value your program produces. When your program returns a zero, then a solution has been found for the unknown variable.

Examples Using SOLVE

Example: Solving for the Dimensions of a Box. Use the following program to evaluate the dimensions of a box ($L \times W \times H - V$). Note that the program uses *recall arithmetic,* which takes less memory than recalling a variable and doing arithmetic as separate operations.

```
B01 LBL B
B02 INPUT L
B03 INPUT W
B04 INPUT H
B05 INPUT V
B06 RCL L
B07 RCL× W
B08 RCL× H
B09 RCL- V
B10 RTN
```

First enter the program labeled B:

Keys:	Display:	Description:
[f] [PRGM]		Starts program entry.
[f] [GTO] [.] [.]	PRGM TOP	Goes to the top of memory (if necessary).

■ LBL/RTN {LBL} B	B01 LBL B	Enters program lines.
■ INPUT L	B02 INPUT L	
■ INPUT W	B03 INPUT W	
■ INPUT H	B04 INPUT H	
■ INPUT V	B05 INPUT V	
RCL L	B06 RCL L	
RCL × W	B07 RCL× W	
RCL × H	B08 RCL× H	
RCL − V	B09 RCL- V	At this point, the X-register will contain the value of the function $L \times W \times H - V$.
■ LBL/RTN {RTN}	B10 RTN	
C		Ends program entry. Displays whatever is in X-register.

Solve for the volume of a box that is 8.5 cm high × 10 cm wide × 25 cm long. Afterward we will use the same function to solve for a different variable.

■ SOLVE/∫ {FN}	FN= _	Prompts for the label of the program that defines the function.
B		Specifies program B.
■ SOLVE/∫ {SOLVE}	SOLVE _	Prompts for the unknown variable.
V	L?*value*	Starts program B; prompts for all data *except V*, the variable being solved.
25 R/S	W?*value*	
10 R/S	H?*value*	
8.5		
R/S	SOLVING V=2,125.0000	The volume is 2,125 cm^3.

Now solve for the length of this box if you change the volume (to 3,000 cm^3), but leave the height and width the same. Remember that you do not need to specify the program label again since we will use the same one that was used last.

[SOLVE/∫] {SOLVE}		Starts program B to solve for *L*. Prompts for unknown variables.
L	W?10.0000	
[R/S]	H?8.5000	
[R/S]	V?2,125.0000	To keep a same value, just press [R/S]. Solves for the length.
3000 [R/S]	L=35.2941	

Example: The Equation of Linear Motion. The equation of motion for a free-falling object is:

$$d = v_0 t + \tfrac{1}{2} g t^2$$

where d is the distance, v_0 is the initial velocity, t is the time, and g is the acceleration due to gravity. Setting the equation equal to zero and simplifying it yields

$$0 = t(v_0 + gt/2) - d.$$

The following program evaluates this function:

```
M01 LBL M
M02 INPUT V
M03 INPUT T
M04 INPUT G
M05 INPUT D
M06 RCL G
M07 2
M08 ÷
M09 RCL× T
N10 RCL+ V
M11 RCL× T
M12 RCL- D
M13 RTN
```

The acceleration due to gravity, g, is included as a variable to allow you to change it for working with different units:

$$g = 9.8 \text{ m/s}^2 = 32.2 \text{ ft/s}^2.$$

Enter the above program (LBL M). Calculate how far an object falls in 5 seconds, starting from rest.

Keys:	Display:	Description:
■ SOLVE/∫ {FN} M	FN= M *(briefly)*	Specifies LBL M for the function.
■ SOLVE/∫ {SOLVE}D	V?3,000.0000	Specifies D as the unknown. Prompts for V as it shows the current value of V (used in the last example).
0 R/S	T? *value*	
5 R/S	G? *value*	
9.8 R/S	D=122.5000	Resulting distance in meters.

Try another calculation using the same equation: how long does it take an object to fall 500 meters? Since v_0 and g are already stored, there is no need to reenter them.

■ SOLVE/∫ {SOLVE}T	V?0.0000	Specifies a different unknown; prompts for V.
R/S	G?9.8000	
R/S	D?122.5000	
500 R/S	T=10.1015	Result in seconds.

Example: Finding the Roots of an Equation. Consider the single-variable equation

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

Rearranging the equation so one side is zero yields

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

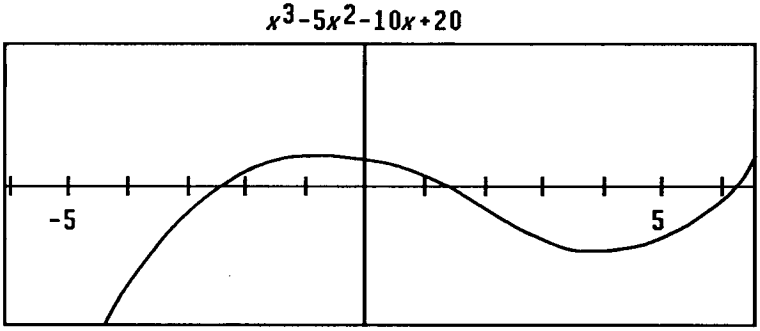
Horner's method (see chapter 5) simplifies this equation to use less memory:

$$x(x(x - 5) - 10) + 20 = 0.$$

The following program evaluates this function:

```
R01 LBL R
R02 RCL X
R03 5
R04 -
R05 RCL× X
R06 10
R07 -
R08 RCL× X
R09 20
R10 +
R11 RTN
```

A plot of this function is:



The plot indicates that there are three roots because the curve crosses the x-axis three times. The calculator can find all three roots if you run SOLVE three times *and supply different initial guesses each time.* (For more information, see "Choosing Initial Guesses for SOLVE.")

Enter the above program (LBL R). The graph shows that the first root is somewhere between $x = -3$ and $x = -2$, the second root is between 1 and 2, and the third root is between 6 and 7. Put each set of guesses in the variable X and in the X-register, then solve for X.

Keys:	Display:	Description:
■ SOLVE/∫ {FN} R		Selects program LBL R.
3 +/- STO X 2 +/-	-3.0000 -2_	First initial guesses.
■ SOLVE/∫ {SOLVE} X	X=-2.4433	Specifies the unknown; returns the first root.
1 STO X 2	1.0000 2_	Second initial guesses.
■ SOLVE/∫ {SOLVE} X	X=1.3416	The second root.
6 STO X 7	6.0000 7_	Third initial guesses.
■ SOLVE/∫ {SOLVE} X	X=6.1017	The third root.

If you did *not* enter any initial guesses, you could get only *one* of these roots. Which one depends on whatever happened to be in the variable X and in the X-register, since the calculator uses these values for initial guesses whether you intend it to or not.

Understanding and Controlling SOLVE

SOLVE uses an iterative (repetitive) procedure to solve for the unknown variable. The procedure starts by substituting two initial guesses for the unknown into the function defined in the program. Based on the result with those two guesses, SOLVE generates another, better guess. Through successive iterations, SOLVE finds a value that makes the function equal to zero.

Some equations are more difficult to solve than others. In some cases, you need to enter initial guesses yourself in order to find the solution. (See "Choosing Initial Guesses for SOLVE," below.) If SOLVE cannot find a real solution, the calculator displays NO ROOT FND.

See appendix C for more information about how SOLVE works.

Verifying the Result

After the SOLVE calculation ends, you can verify that the result is indeed a root of $f(x)$ by reviewing the values left in the stack:

- The display (the X-register) and the variable itself contain the solution (root) for the unknown; that is, the value that makes $f(x) = 0$.
- The Y-register (press [R↓] to view Y) contains the previous estimate for the root. This number should be the same as the value in the X-register. If it is *not*, then the root given was only an *approximation* to the root, and the values in the X- and Y-register bracket the root. These bracketing numbers should be close together.
- The Z-register (press [R↓] again to view Z) contains the value of $f(x)$ at the value given in the X-register. For an exact root, this should be zero. If it is not zero, then the root given was only an *approximation*, and this number should be close to zero.

If a calculation ends with NO ROOT FND, this means that the calculation could not converge on a root, so the values in the X- and Y-registers are probably not close together. (You can see the value in the X-register—the final estimate of the root—by pressing [C] or [←] to clear the message.) These two values bracket the interval that was last searched for the root. The Z-register contains the value of $f(x)$ at the final estimate of the root. This value should not be close to zero.

Interrupting the SOLVE Calculation

To halt the calculation, press [C] or [R/S]. The current best estimate of the root is in the unknown variable; use ■[VIEW] to view it without disturbing the stack. To resume the calculation, press [R/S].*

* Pressing [XEQ], [GTO], or {RTN} cancels the SOLVE operation. In this case, start the program over rather than resuming it.

Choosing Initial Guesses for SOLVE

The two initial guesses come from:

- The number currently stored in the unknown variable.
- The number in the X-register (the display).

These sources are used for guesses *whether you enter guesses or not*. If you enter only one guess and store it into the variable, then the second guess will be the same value since the display also holds the number you just stored into the variable. Entering your own guesses has these advantages:

- By narrowing the range of the search, guesses can reduce the time required to find a solution.
- If there is more than one mathematical solution, the guesses can direct the SOLVE procedure to the desired answer or range of answers. For example, the equation of motion

  $$d = d_0 + v_0 t + \tfrac{1}{2} g t^2$$

  can have two solutions for t. You can direct the answer to the only meaningful one ($t > 0$) by entering appropriate guesses.*
- If an equation does not allow certain values for the unknown, guesses can prevent these values from occurring. For example, the equation

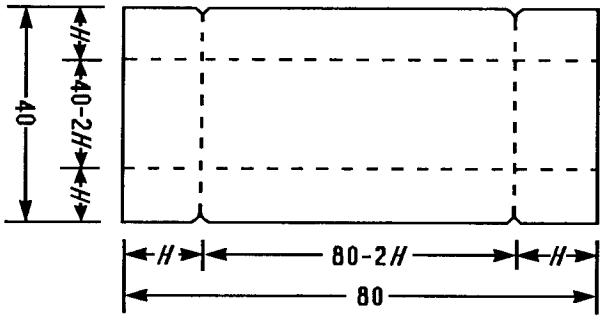
  $$y = t + \log x$$

  results in an error if $x \leq 0$ (`LOG‹0›`, `LOG‹NEG›`).

The example in the previous section ("Finding the Roots of an Equation") used guesses to find three solutions to one equation. Here is another example that examines the dimensions of a box (as does the example on page 113) but with more restrictions.

* The example using this equation on page 116 did not need to enter guesses before solving for t because in the first part of the example we stored a value for T and solved for D. The value that was left in T was a good (realistic) one, and it was used as a guess when solving for T.

Example: Folding a Box. Using a rectangular piece of sheet metal 40 centimeters by 80 centimeters, form an open-top box having a volume of 7500 cubic centimeters. You need to find out the height of the box (that is, the amount to be folded up along each of the four sides) that gives the specified volume. *A taller box is preferred to a shorter one.*



If H is the height, then the length of the box is $(80 - 2H)$ and the width is $(40 - 2H)$. The volume, V, is:

$$V = (80 - 2H) \times (40 - 2H) \times H$$

and the function equal to zero is

$$\begin{aligned} 0 &= (80 - 2H) \times (40 - 2H) \times H - V \\ &= 4H\,[(40 - H)\,(20 - H)] - V \end{aligned}$$

One program to define this function would be:

```
V01 LBL V
V02 INPUT H
V03 INPUT V
V04 40
V05 RCL- H
V06 20
V07 RCL- H
V08 ×
V09 4
V10 ×
V11 RCL× H
V12 RCL- V
V13 RTN
```

It seems reasonable that either a tall, narrow box or a short, flat box could be formed having the desired volume. Because the taller box is preferred, larger initial estimates of the height are reasonable. However, heights greater than 20 centimeters are not physically possible because the metal sheet is only 40 centimeters wide. Initial estimates of 10 and 20 centimeters are therefore appropriate.

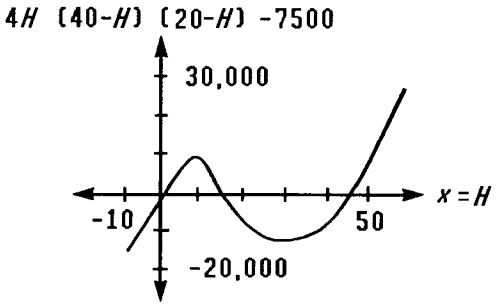
Keys:	**Display:**	**Description:**
■ [SOLVE/∫] {FN} V		Selects program V as the function to solve.
10 [STO] H 20	10.0000 20_	Stores upper and lower limits.
■ [SOLVE/∫] {SOLVE} H	 V?*value*	Prompts for volume.
7500 [R/S]	SOLVING H=15.0000	This is the desired height.

Now check the quality of this solution—that is, whether it provided an exact root—by looking at the values of the previous estimate (in the Y-register) and $f(x)$ at the root (in the Z-register).

R↓	15.0000	This value from the Y-register is the estimate made just prior to the final result. Since it is the same as the solution, the solution is an exact root...
R↓	0.0000	...and, as this value from the Z-register shows, $f(x) = 0$ at the root.

The dimensions of the desired box are 50 × 10 × 15 cm. If you ignored the upper limit on the height (20 cm) and used initial estimates of 30 and 40 cm, you would obtain a height of 42.0256 cm—a root that is physically meaningless. If you used small initial estimates such as 0 and 10 cm, you would obtain a height of 2.9744 cm—producing an undesirably short, flat box.

Using Graphs to Select Initial Guesses. As an aid to understanding the behavior of a particular function, you can graph it. Use your program routine to evaluate the function for several values of the unknown. For each point on the graph, store the value for the *x-coordinate* in the variable, and then obtain the corresponding value for the *y-coordinate* by pressing XEQ *label*. For the problem above, you would always set $V = 7500$ and vary the value of H to produce different values for the function. The plot of this function looks like this:



Using SOLVE in a Program

You can use the SOLVE operation as part of a program. If appropriate, include or prompt for initial guesses (into the unknown variable and into the X-register) before executing the SOLVE *variable* instruction. The two instructions for solving an equation for an unknown variable appear in the program as:

```
FN= label
SOLVE variable
```

Labeling Output. The *programmed* SOLVE instruction does not produce a labeled display (*variable=value*) since this might not be the significant output for your program (that is, you might want to do further calculations with this number before displaying it). If you *do* want this result displayed, add a VIEW *variable* instruction after the SOLVE instruction.

Conditional Execution if No Solution. If no solution is found for the unknown variable, then the next program line is skipped (in accordance with the "Do if True" rule, explained in chapter 6). The program should then handle the case of not finding a root, such as by choosing new initial estimates or changing an input value.

Example: Time Value of Money. The "Time Value of Money" program in chapter 14 solves loan and savings problems by solving for an unknown in the given TVM equation. This equation is defined as a function in routine T, which relates the variables for present balance, future balance, payment, interest rate, and number of payments.

Given any four of these variables, the SOLVE instruction (line L04) finds the solution for the fifth one:

`L01 LBL L`	
`L02 STO i`	Stores an index value that indicates which variable had been specified as the unknown.
`L03 FN= T`	Selects the function defined in program T.
`L04 SOLVE(i)`	Solves for the indicated unknown variable in program T.
`L05 VIEW(i)`	Displays the resulting solution.
`L06 GTO(i)`	Returns to the initializing subroutine to prepare for another calculation.

This SOLVE operation works fine without the user supplying initial guesses.

Limitations. The SOLVE instruction cannot call a routine that contains another SOLVE instruction; that is, it cannot be used recursively (`SOLVE(SOLVE` error). Nor can SOLVE call a routine that contains a `FN=` *label* instruction (`SOLVE ACTIVE` error). SOLVE cannot call a routine that contains an ∫FN instruction (`SOLVE(∫FN` error), just as ∫FN cannot call a routine that contains a SOLVE instruction (`∫(SOLVE` error).

The `SOLVE` *variable* instruction in a program uses one of the seven pending subroutine returns in the calculator. (Refer to "Nested Subroutines" in chapter 6.)

For More Information

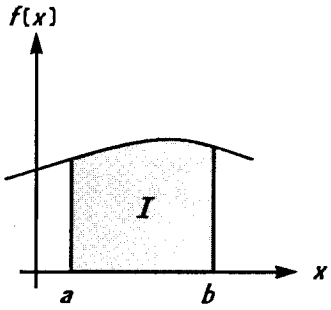
This chapter gives you instructions for solving for unknowns or roots over a wide range of applications. Appendix C contains more detailed information about how the algorithm for SOLVE works, how to interpret results, what happens when no solution is found, and conditions that can cause incorrect results.

8

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 *a* to *b*, then the integral can be expressed mathematically as

$$I = \int_a^b f(x)\, dx.$$



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

The ∫FN operation integrates a specified function with respect to a specified variable.* The function must be defined beforehand in a labeled program, and it may have more than one variable.

* ∫FN works only with real numbers.

Using Integration (∫FN)

To integrate a function:

1. Enter a program that defines the integrand's function. (See "Writing Programs for ∫FN" below.)
2. Select the program that defines the function to integrate: press
   [SHIFT] [SOLVE/∫] {FN} *label*.
3. Enter the limits of integration: key in the *lower limit* and press [ENTER], then key in the *upper limit*.
4. Select the variable of integration: press
   [SHIFT] [SOLVE/∫] {∫FN} *variable*.

   This starts the calculation.

This operation uses far more memory than any other operation in the calculator. If executing {∫FN} causes a MEMORY FULL message, refer to appendix B.

You can halt a running integration calculation by pressing [C] or [R/S]. (However, no information about the integration is available until the calculation finishes normally.) To resume the calculation, press [R/S].*

Accuracy. The display format setting affects the level of accuracy assumed for your function and used for the result. Integration is more precise but takes *much* longer in the {ALL} and higher {FX}, {SC}, and {EN} settings. The *uncertainty* of the result ends up in the Y-register, pushing the limits of integration up into the T- and Z-registers. For more information, see "Accuracy of Integration," page 131.

Results. The X-register contains the integral, the Y-register the uncertainty, the Z-register the upper limit, and the T-register the lower limit. (The variable of integration contains an immaterial value.)

* Pressing [XEQ], [GTO], or {RTN} cancels the ∫FN operation. In this case, start the operation over rather than resuming it.

To integrate the same function with different information: Skip the first two steps above. If using the same limits of integration, press [R↓] [R↓] to bring them back into the X- and Y-registers. (If *not* using the same limits, repeat step 3.) Then execute ■ [SOLVE/∫] {∫FN} *variable*. (To work another problem using a *different* function, start over with a different program for the function.)

Writing Programs for ∫FN

To write a program defining the integrand's function:

1. Begin with a label so that the program can be called by ∫FN.
2. Include an initial INPUT instruction for each variable, including the variable of integration. (If there is only one variable in the function, you can omit the INPUT instruction.)*
3. Enter the instructions to define the function. Use a RCL instruction any place a variable's value is needed for a calculation.
4. End the program with a RTN. The program should end with the value of the function in the X-register.

Examples Using ∫FN

Example: Bessel Function. The Bessel function of the first kind of order 0 can be expressed as

$$J_0(x) = {}^1/\pi \int_0^\pi \cos(x \sin t)\, dt.$$

Find the Bessel function for x-values of 2 and 3.

* The INPUT instructions are useful for multi-variable functions. Since the INPUT for the variable of integration (integrand) is ignored, you can write *one* program that contains INPUT instructions for *all* variables, which you can then use no matter which variable you specify as the variable of integration.

This program evaluates the function $f(t) = \cos(x \sin t)$:

```
J01 LBL J
J02 RAD
J03 INPUT X
J04 INPUT T
J05 RCL T
J06 SIN
J07 RCL× X
J08 COS
J09 RTN
```

These keystrokes enter the program:

Keys:	Display:	Description:
[f] [PRGM] [f] [GTO] [.] [.]	PRGM TOP	Starts program entry; places program pointer at the top of memory.
[f] [LBL/RTN] {LBL} J	J01 LBL J	Enters program.
[f] [MODES] {RD}	J02 RAD	
[f] [INPUT] X	J03 INPUT X	
[f] [INPUT] T	J04 INPUT T	
[RCL] T [SIN]	J06 SIN	
[RCL] [×] X [COS]	J08 COS	At this point, the X-register will contain the value of the function.
[f] [LBL/RTN] {RTN}	J09 RTN	
[C]		Ends program entry.

Now integrate this function with respect to t from zero to π; $x = 2$.

[f] [SOLVE/∫] {FN} J	FN= _	Selects routine J for the function.
0 [ENTER] [f] [π]	3.1416	Enters limits of integration (lower first).
[f] [SOLVE/∫] {∫FN} T	∫FN d _ X?*value*	Specifies T as the variable of integration. Prompts for the value of x.

2 [R/S]	∫=0.7034	$x = 2$. Starts integrating and produces the result for $\int_0^{\pi} f(t)$.

To complete the calculation, remember to multiply the value of the integral by the constant $(1/\pi)$ outside the integral. (You could also include this multiplication as part of the program.)

■[π] [÷]	0.2239	Final result for $J_0(2)$.

Now calculate $J_0(3)$ with the same limits of integration. You don't have to respecify the function (routine J), but you must respecify the limits of integration $(0, \pi)$ since they were pushed off the stack by the subsequent division by π.

0 [ENTER] ■[π]	3.1416	Displays upper limit.
■[SOLVE/∫] {∫FN} T	X?2.0000	Starts integration; prompts for x.
3 [R/S]	∫=-0.8170	Integral of $f(t)$.
■[π] [÷]	-0.2601	Result for $J_0(3)$.

Example: Sine Integral. Certain problems in communications theory (for example, pulse transmission through idealized networks) require calculating an integral (sometimes called the *sine integral*) of the form

$$Si(t) = \int_0^t \left(\frac{\sin x}{x}\right) dx.$$

Find $Si(2)$.

Key in the following program to evaluate the function $f(x) = (\sin x) \div x$.*

* If the calculator attempted to evalutate this function at $x = 0$, the lower limit of integration, an error (DIVIDE BY 0) would result. However, the integration algorithm normally does *not* evaluate functions at either limit of integration, unless the endpoints of the interval of integration are extremely close together or the number of sample points is extremely large.

```
S01 LBL S
S02 RAD
S03 RCL X
S04 SIN
S05 RCL÷ X
S06 RTN
```

Now integrate this function with respect to x (that is, X) from zero to 2 ($t = 2$).

Keys:	Display:	Description:
■ [SOLVE/∫] {FN} S		Selects routine S for the function.
0 [ENTER] 2	2_	Enters limits of integration (lower first).
■ [SOLVE/∫] {∫FN}X	∫=1.6054	Result for $Si(2)$.

Accuracy of Integration

Since the calculator cannot compute the value of an integral exactly, it *approximates* it. The accuracy of this approximation depends on the accuracy of the integrand's function itself, as calculated by your program.* This is affected by round-off error in the calculator and the accuracy of the empirical constants.

* Integrals of functions with certain characteristics such as spikes or very rapid oscillations *might* be calculated inaccurately, but the likelihood is very small. The general characteristics of functions that can cause problems, as well as techniques for dealing with them, are discussed in appendix D.

Specifying Accuracy

The display format's setting determines the *precision* of the integration calculation: the greater the number of digits displayed, the greater the precision of the calculated integral (and the greater the time required to calculate it). The fewer the number of digits displayed, the faster the calculation, but the calculator will presume that the function is accurate to only the number of digits specified in the display format.

To specify the *accuracy* of the integration, set the display format so that the display shows *no more than* the number of digits that you consider accurate *in the integrand's values*. This same level of accuracy and precision will be reflected in the result of integration.

Interpreting Accuracy

After calculating the integral, the calculator places the estimated *uncertainty* of that integral's result in the Y-register. Press [x≷y] to view the value of the uncertainty.

For example, if the integral $Si(2)$ is 1.6054 ± 0.0001, then 0.0001 is its uncertainty.

Example: Specifying Accuracy. With the display format set to SCI 2, calculate the integral in the expression for $Si(2)$ (from the previous example).

Keys:	Display:	Description:
[DISP] {SC}2	1.61E0	Sets scientific notation with two decimal places, specifying that the function is accurate to two decimal places.
0 [ENTER] 2	2_	Limits of integration.
[SOLVE/∫] {FN} S	2.00E0	Selects routine S for the function.
[SOLVE/∫] {∫FN} X	∫=1.61E0	Integral approximated to two decimal places.

[x≷y]	1.00E-3	The uncertainty of the approximation of the integral.

The integral is 1.61 ± 0.00100. Since the uncertainty would not affect the approximation until its third decimal place, you can consider all the displayed digits in this approximation to be accurate.

If the uncertainty of an approximation is larger than what you choose to tolerate, you can increase the number of digits in the display format and repeat the integration (provided that $f(x)$ is still calculated accurately to the number of digits shown in the display). In general, the uncertainty of an integration calculation decreases by a factor of 10 for each additional digit specified in the display format.

Example: Changing the Accuracy. For the integral of $Si(2)$ just calculated, specify that the result be accurate to four decimal places instead of only two.

Keys:	**Display:**	**Description:**
[■] [DISP] {SC} 4	1.0000E-3	Specifies accuracy to four decimal places. The uncertainty from the last example is still in the display.
[R↓] [R↓]	2.0000E0	Rolls down the limits of integration from the Z- and T-registers into the X- and Y-registers.
[■] [SOLVE/∫] {∫FN} X	∫=1.6054E0	Result.
[x≷y]	1.0000E-5	Note that the uncertainty is about 1/100 as large as the uncertainty of the SCI 2 result calculated previously.
[■] [DISP] {FX} 4		Restores FIX 4 format.

This uncertainty indicates that the result *might* be correct to only four decimal places. In reality, this result is accurate to *seven* decimal places when compared with the actual value of this integral. Since the uncertainty of a result is calculated conservatively, *the calculator's approximation in most cases is more accurate than its uncertainty indicates.*

For more information, see appendix D.

Using Integration in a Program

Integration can be executed from a program. Remember to include or prompt for the limits of integration before executing the integration, and remember that accuracy and execution time are controlled by the display format at the time the program runs. The two integration instructions appear in the program as:

FN= *label*
∫FN d *variable*

Labeling Output. The *programmed* ∫FN instruction does not produce a labeled display (∫=*value*), since this might not be the significant output for your program (that is, you might want to do further calculations with this number). If you *do* want this result displayed, add a PSE (■ LBL/RTN {PSE}) or STOP (R/S) instruction to display the result in the X-register after the ∫FN instruction.

Example: Normal Distribution. The "Normal and Inverse-Normal Distributions" program on page 215 in part 4 includes an integration of the equation of the normal density function,

$$\frac{1}{S\sqrt{2\pi}}\int_M^D e^{-\left(\frac{D-M}{S}\right)^2 \div 2}\,dD.$$

This function is defined in routine F:

```
F01 LBL F
F02 RCL D
F03 RCL- M
F04 RCL÷ S
F05 x²
F06 2
F07 ÷
F08 +/-
F09 eˣ
F10 RTN
```

Other routines prompt for the known values and do the other calculations to find $Q(D)$, the upper-tail area of a normal curve. The integration itself is set up and executed from routine Q:

```
Q01 LBL Q
```

`Q02 RCL M`	Recalls the lower limit of integration.
`Q03 RCL X`	Recalls the upper limit of integration. ($X = D$.)
`Q04 FN= F`	Specifies the function defined by LBL F for integration.
`Q05 ∫FN d D` ⋮	Integrates the normal function for the variable D.

Limitations. The `∫FN` *variable* instruction cannot call a routine that contains another ∫FN instruction; that is, it cannot be used recursively, so you cannot calculate multiple integrals (`∫(∫FN)` error). Nor can ∫FN call a routine that contains a `FN=` *label* instruction (`∫FN ACTIVE` error). ∫FN cannot call a routine that contains a SOLVE instruction (`∫(SOLVE)` error), just as SOLVE cannot call a routine that contains an integration instruction (`SOLVE(∫FN)` error).

The `∫FN d` *variable* instruction in a program uses one of the seven pending subroutine returns in the calculator. (Refer to "Nested Subroutines" in chapter 6.)

For More Information

This chapter gives you instructions for using integration in the HP–32S over a wide range of applications. Appendix D contains more detailed information about how the algorithm for integration works, conditions that could cause incorrect results, conditions that prolong calculation time, and obtaining the current approximation to an integral.

9

Operations With Complex Numbers

The HP-32S can use complex numbers in the form

$$x + iy.$$

It has operations for complex arithmetic ($+$, $-$, $\times$, $\div$), complex trigonometry (sin, cos, tan), and the mathematics functions $-z$, $1/z$, $z_1^{z_2}$, ln z, and e^z (where z_1 and z_2 are complex numbers).

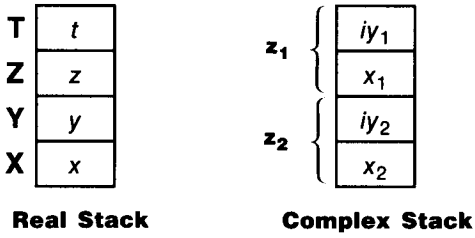
Complex numbers in the HP-32S are handled by entering each part (imaginary and real) of a complex number as a separate entry. To enter two complex numbers, you enter four separate numbers. To do a complex operation, press ■ CMPLX before the operator. For example, to do

$$(2 + i4) + (3 + i5),$$

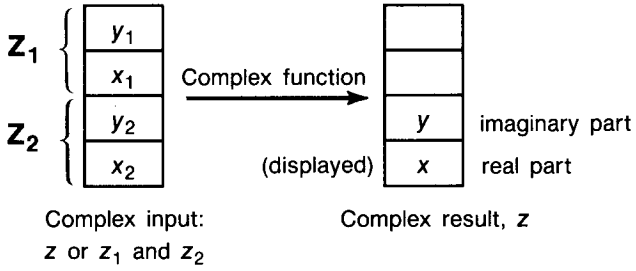
press 4 ENTER 2 ENTER 5 ENTER 3 ■ CMPLX +. The result is $5 + i9$. (Press x≷y to see the imaginary part.)

The Complex Stack

The complex stack is really the regular memory stack split into two double registers for holding two complex numbers, $z_{1x} + iz_{1y}$ and $z_{2x} + iz_{2y}$:



Since the imaginary and real parts of a complex number are entered and stored separately, you can easily work with or alter either part by itself.



Always enter the imaginary part (the *y*-part) *of a number first.* The real portion of the result (z_x) is displayed; press [x≷y] to view the imaginary portion (z_y).

Complex Operations

Use the complex operations as you do real operations, but precede the operator with ■ CMPLX.

To do an operation with one complex number:

1. Enter the complex number z, composed of $x + iy$, by keying in y ENTER x.
2. Select the complex function:

Functions for One Complex Number, z

To Calculate:	Press:
Change sign, $-z$	■ CMPLX +/-
Inverse, $1/z$	■ CMPLX 1/x
Natural log, ln z	■ CMPLX LN
Natural antilog, e^z	■ CMPLX e^x
Sin z	■ CMPLX SIN
Cos z	■ CMPLX COS
Tan z	■ CMPLX TAN

To do an arithmetic operation with two complex numbers:

1. Enter the first complex number, z_1 (composed of $x_1 + iy_1$), by keying in y_1 ENTER x_1 ENTER. (For $z_1^{z_2}$, key in the base part, z_1, first.)
2. Enter the second complex number, z_2, by keying in y_2 ENTER x_2. (For $z_1^{z_2}$, key in the superscript, z_2, second.)
3. Select the arithmetic operation:

Arithmetic With Two Complex Numbers, z_1 and z_2

To Calculate:	Press:
Addition, $z_1 + z_2$	[CMPLX] [+]
Subtraction, $z_1 - z_2$	[CMPLX] [−]
Multiplication, $z_1 \times z_2$	[CMPLX] [×]
Division, $z_1 \div z_2$	[CMPLX] [÷]
Power function, $z_1^{z_2}$	[CMPLX] [y^x]

Examples. Here are some examples of trigonometry and arithmetic with complex numbers:

Evaluate sin $(2 + i3)$.

Keys:	Display:	Description:
3 [ENTER] 2		Real part of result.
[CMPLX] [SIN]	9.1545	
[x≷y]	-4.1689	Result is $9.1545 - i4.1689$.

Evaluate the expression

$$z_1 \div (z_2 + z_3),$$

where $z_1 = 23 + i13$, $z_2 = -2 + i$, $z_3 = 4 - i3$.

Since the stack can retain only two complex numbers at a time, perform the calculation as

$$z_1 \times [1 \div (z_2 + z_3)].$$

Keys:	Display:	Description:
1 [ENTER] 2 [+/−] [ENTER] 3 [+/−] [ENTER] 4 ■ [CMPLX] [+]	2.0000	Adds $z_2 + z_3$; displays real part.
■ [CMPLX] [1/x]	0.2500	$1 \div (z_2 + z_3)$.
13 [ENTER] 23 ■ [CMPLX] [×]	2.5000	$z_1 \div (z_2 + z_3)$.
[x≷y]	9.0000	Result is $2.5 + i9$.

Evaluate $(4 - i2/5)(3 - i2/3)$. Do not use complex operations when calculating just one part of a complex number.

Keys:	Display:	Description:
2 [ENTER] 5 [÷] [+/−]	-0.4000	Calculates imaginary part using real operations.
4 [ENTER]	4.0000	Enters real part of first complex number.
2 [ENTER] 3 [÷] [+/−]	-0.6667	Calculates imaginary part of second complex number.
3 ■ [CMPLX] [×]	11.7333	Completes entry of second number and then multiplies the two complex numbers.
[x≷y]	-3.8667	Result is $11.7333 - i3.8667$.

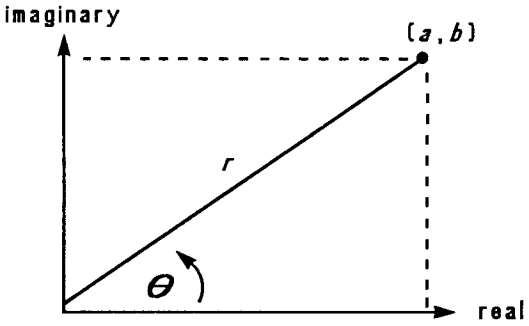
Evaluate $e^{z^{-2}}$, where $z = (1 + i)$. Use ■ [CMPLX] [y^x] to evaluate z^{-2}; enter -2 as $-2 + i0$.

Keys:	Display:	Description:
1 [ENTER] 1 [ENTER] 0 [ENTER] 2 [+/-]		Intermediate result of $(1 + i)^{-2}$.
[■] [CMPLX] [y^x]	0.0000	
[■] [CMPLX] [e^x]	0.8776	Real part of final result.
[x≷y]	-0.4794	Final result is $0.8776 - i0.4794$.

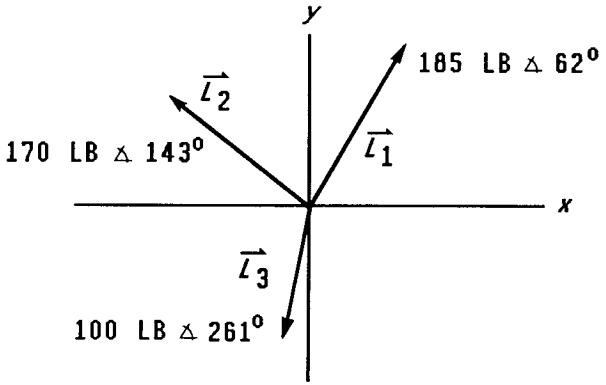
Using Numbers in Polar Notation

Many applications use real numbers in *polar* form or *phasor* notation. These forms use pairs of numbers, as do complex numbers, so you can do arithmetic with these numbers by using the complex operations. Since the HP-32S's complex operations work on numbers in *rectangular* form, convert polar form to rectangular form (using [■] [P↔RECT]) before executing the complex operation, then convert the result back to polar form.

$$a + ib = r(\cos\theta + i\sin\theta) = re^{i\theta}$$
$$= r \angle \theta \quad \text{Polar or phasor form}$$



Example: Vector Addition. Add the following three loads. You will first need to convert the polar coordinates to rectangular coordinates.



Keys:	Display:	Description:
■ MODES {DG}		Sets Degrees mode.
62 ENTER 185 ■ P↔RECT {θ,r→y,x}	86.8522	Enters $\mathbf{L}_1$ and converts it to rectangular form.
143 ENTER 170 ■ P↔RECT {θ,r→y,x}	-135.7680	Enters and converts $\mathbf{L}_2$.
■ CMPLX +	-48.9158	Adds vectors.
261 ENTER 100 ■ P↔RECT {θ,r→y,x}	-15.6434	Enters and converts $\mathbf{L}_3$.
■ CMPLX +	-64.5592	Adds $\mathbf{L}_1 + \mathbf{L}_2 + \mathbf{L}_3$.
■ P↔RECT {y,x→θ,r}	178.9372	Converts vector back to polar form; displays r.
x≷y	111.1489	Displays θ.

10

Base Conversions and Arithmetic

The BASE menu (■ BASE) lets you change the number base used for entering numbers and other operations (including programming). Changing bases also converts the *displayed* number to the new base.

The BASE Menu

Menu Label	Description
{DEC}	*Decimal mode*. No annunciator. Converts numbers to base 10. Numbers have integer and fractional parts.
{HX}	*Hexadecimal mode*. **HEX** annunciator on. Converts numbers to base 16; uses integers only. The top-row keys become digits A through F.
{OC}	*Octal mode*. **OCT** annunciator on. Converts numbers to base 8; uses integers only. The 8, 9, and unshifted top-row keys are inactive.
{BN}	*Binary mode*. **BIN** annunciator on. Converts numbers to base 2; uses integers only. Digit keys other than 0 and 1, and the unshifted top-row functions are inactive. If a number is longer than 12 digits, then the outer top-row keys ($\sqrt{x}$ and Σ+) are active for viewing windows. (See "Windows for Long Binary Numbers" in this chapter.)

Examples: Converting the Base of a Number. The following keystrokes do various base conversions.

Convert 125.99_{10} to hexadecimal, octal, and binary numbers.

Keys:	Display:	Description:
125.99 ■ [BASE] {HX}	7D	Converts just the integer part (125) of the decimal number to base 16 and displays this value.
■ [BASE] {OC}	175	Base 8.
■ [BASE] {BN}	1111101	Base 2.
■ [BASE] {DEC}	125.9900	Restores base 10; the original decimal value has been preserved, including its fractional part.

Convert $24FF_{16}$ to binary base. The binary number will be more than 12 digits (the maximum display) long.

■ [BASE] {HX} 24FF	24FF_	Use the [Σ+] key to type "F".
■ [BASE] {BN}	010011111111	The entire binary number does not fit. The ← annunciator indicates that the number continues to the left; the ▼ annunciator points to [√x].
[√x]	10	Displays the rest of the number. The full number is 10010011111111_2.
[Σ+]	010011111111	Displays the first 12 digits again.
■ [BASE] {DEC}	9,471.0000	Back to base 10.

Arithmetic in Bases 2, 8, and 16

You can perform arithmetic operations using [+], [−], [×], and [÷] in any base.* Arithmetic in bases 2, 8, and 16 is in 2's complement form and uses integers only:

- If a number has a fractional part, only the integer part is used for an arithmetic calculation.
- The result of an operation is always an integer (any fractional portion is truncated).

Whereas conversions change only the displayed number and not the number in the X-register, *arithmetic does* alter the number in the X-register.

If the result of an operation cannot be represented in 36 bits, the display shows OVERFLOW and then the largest positive or negative number possible.

Examples. Here are some examples of arithmetic in Hexadecimal, Octal, and Binary modes:

$$12F_{16} + E9A_{16} = ?$$

Keys:	Display:	Description:
■ [BASE] {HX}		Sets base 16; **HEX** annunciator on.
12F [ENTER] E9A [+]	FC9	Result.

* The only function keys that are actually deactivated outside of Decimal mode are [√x], [e^x], [LN], [y^x], [1/x], [Σ+]. However, you should realize that most operations other than arithmetic will not produce meaningful results since the fractional parts of numbers are truncated.

$$7760_8 - 4326_8 = ?$$

■ BASE {OC}	7711	Sets base 8; **OCT** annunciator on. Converts displayed number to octal.
7760 ENTER 4326 −	3432	Result.

$$100_8 \div 5_8 = ?$$

100 ENTER 5 ÷	14	Integer part of result.

$$5A0_{16} + 1001100_2 = ?$$

■ BASE {HX} 5A0	5A0_	Sets base 16; **HEX** annunciator on.
■ BASE {BN} 1001100	1001100_	Changes to base 2; **BIN** annunciator on. This terminates digit entry, so no ENTER is needed between the numbers.
+	10111101100	Result in binary base.
■ BASE {HX}	5EC	Result in hexadecimal base.
■ BASE {DC}	1,516.0000	Restores decimal base.

The Representation of Numbers

Although the *display* of a number is converted when the base is changed, its stored form is not modified, so decimal numbers are not truncated—until they are used in arithmetic calculations.

When a number appears in hexadecimal, octal, or binary base, it is shown as a right-justified integer with up to 36 bits (12 octal digits or 9 hexadecimal digits). Leading zeros are not displayed, but they are important because they indicate a positive number. For example, the binary representation of 125_{10} is displayed as:

1111101

which is the same as these 36 digits:

000000000000000000000000000001111101

Negative Numbers

The leftmost (most significant or "highest") bit of a number's binary representation is the sign bit; it is set (1) for negative numbers. If there are (undisplayed) leading zeros, then the sign bit is 0 (positive). A negative number is the 2's complement of its positive binary number.

Keys:	Display:	Description:
546 ■ BASE {HX}	222	Enters a positive, decimal number; then converts it to hexadecimal.
+/-	FFFFFDDE	2's complement (sign changed).
■ BASE {BN}	110111011110	Binary version; ← indicates more digits.
√x √x	111111111111	Displays the leftmost window; the number is negative since the highest bit is 1.
■ BASE {DEC}	-546.0000	Negative decimal number.

Range of Numbers

The 36-bit word size determines the range of numbers that can be represented in hexadecimal (9 digits), octal (12 digits), and binary bases (36 digits), and the range of decimal numbers (11 digits) that can be converted to these other bases.

Range of Numbers for Base Conversions

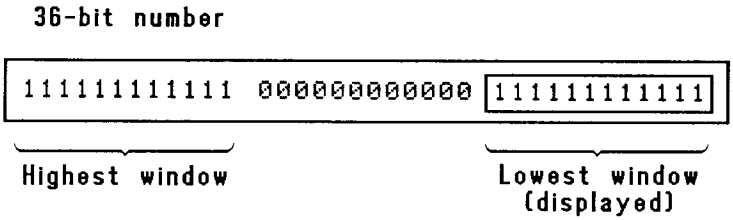
Base	Positive Integer of Largest Magnitude	Negative Integer of Largest Magnitude
Hexadecimal	7FFFFFFFF	800000000
Octal	377777777777	400000000000
Binary	011111111111111111 111111111111111111	100000000000000000 000000000000000000
Decimal	34,359,738,367	−34,359,738,368

When you key in numbers, the calculator will not accept more than the maximum number of digits for each base. For example, if you attempt to key in a 10-digit hexadecimal number, digit entry halts and the ⚠ annunciator appears.

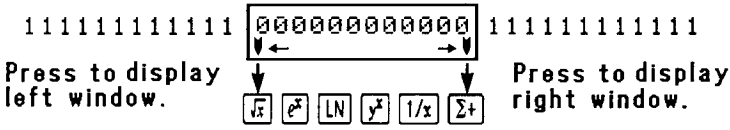
If a number entered in decimal base is outside the range given above, then it produces the message TOO BIG in the other base modes. Any operation using TOO BIG causes an overflow condition, which substitutes the largest positive or negative number possible for the too-big number.

Windows for Long Binary Numbers

The longest binary number can have 36 digits—three times as many digits as fit in the display. Each 12-digit display of a long number is called a *window*.



When a binary number is larger than the 12 digits, the ← or → annunciator (or both) appears, indicating in which direction the additional digits lie. Press the indicated key ([√x] or [Σ+]) to view the obscured window.



SHOWing Partially Hidden Numbers

The ■ [VIEW] and ■ [INPUT] functions work with non-decimal numbers as they do with decimal numbers. However, if the full octal or binary number does not fit in the display, the *leftmost* digits are replaced with ellipses (...). Press ■ [SHOW] to view the digits obscured by the A=... or A?... labels.

Keys:	Display:	Description:
■ [BASE] {OC}		Enters a large octal number.
123456712345	23456712345_	
[STO] A	123456712345	

■ VIEW A	A=...456712345	Drops leftmost three digits.
■ SHOW *(hold)*	123456712345	Shows all digits.

Programming With BASE

You can program instructions to change the base mode using ■ BASE. These settings work in programs just as they do as functions used from the keyboard. This allows you to write programs that accept numbers in any of the four bases, do arithmetic in any base, and display results in any base.

When writing programs that use numbers in a base other than 10, set the base mode both *as the current setting* for the calculator and *in the program* (as an instruction).

Selecting a Base Mode in a Program

Insert a BIN, OCT, or HEX instruction into the beginning of the program. You should usually include a DEC instruction at the end of the program so that the calculator's setting will revert to Decimal mode when the program is done.

An instruction in a program to change the base mode will determine how input is interpreted and how output looks *during and after program execution,* but it does *not* affect the program lines as you enter them.

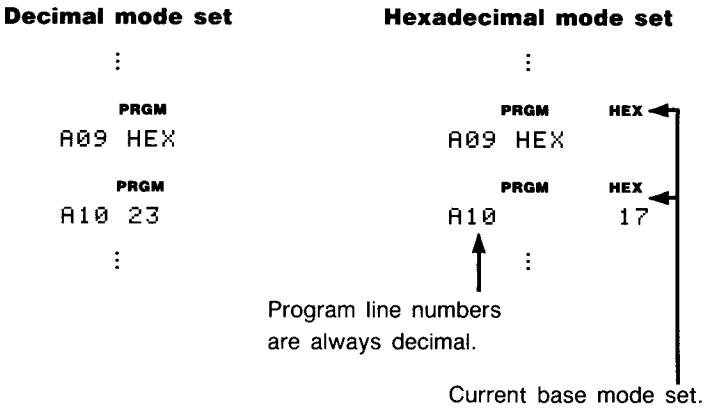
The SOLVE and ∫FN operations automatically set DEC mode.

Numbers Entered in Program Lines

Before starting program entry, set the base mode. The current setting for the base mode determines the base of the numbers that are entered into program lines. The display of these numbers changes when you change the base mode.

Program line numbers always appear in base 10.

An annunciator tells you which base is the current setting. For instance, compare the program lines below in the left and right columns. Notice that the hexadecimal number, like all non-decimal numbers, is right-justified.

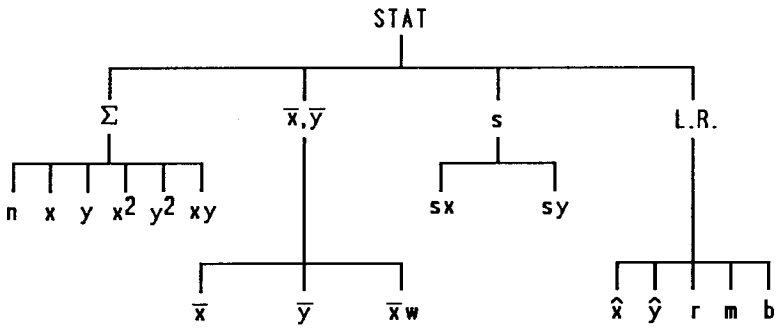


11

Statistical Operations

The STAT (*statistics*) menu provides functions to statistically analyze a set of one- or two-variable data.

- One-variable data: mean and standard deviation.
- Two-variable data (x,y): linear regression and linear estimation ($\hat{x}$ and $\hat{y}$).
- Weighted mean (x weighted by y).
- Summation statistics: n, Σx, Σy, Σx^2, Σy^2, and Σxy.



Entering Statistical Data ([Σ+], ■[Σ−])

One- and two-variable statistical data are entered in similar fashion. The data values are accumulated as summation statistics in six *statistics registers*, whose values are displayed under ■[STAT] {Σ}.

Entering One-Variable Data

1. Press [f] [CLEAR] {Σ} to clear previous statistical data.
2. Key in each x-value and press [Σ+].
3. The display shows n, the number of statistical data values now accumulated.*

To recall a value to the display immediately after it has been entered, press [f] [LASTx].

Entering Two-Variable Data

When your data consist of two variables, x is the *independent variable* and y is the *dependent variable*. Remember to enter an (x, y) pair in reverse order so that y ends up in the Y-register and x in the X-register.

1. Press [f] [CLEAR] {Σ} to clear previous statistical data.
2. Key in the y-value *first* and press [ENTER].
3. Key in the corresponding x-value and press [Σ+].
4. The display shows n, the number of statistical data pairs now accumulated.
5. Continue entering x,y-pairs. The n-value is updated with each entry.

To recall an x-value to the display immediately after it has been entered, press [f] [LASTx].

* This procedure actually enters two variables into the statistics registers because the value already in the Y-register is accumulated as the y-value. For this reason, the calculator will do linear regression and show you values based on y even when you have entered only x-data—or even if you have entered an unequal number of x- and y-values. No error occurs, but the results are obviously not meaningful.

Correcting Errors in Data Entry

If you make a mistake in entering statistical data, delete the incorrect data and add the correct data. Even if only one value of an x, y-pair is incorrect, you must delete and then reenter both values.

To correct statistical data:

1. Reenter the incorrect data, but instead of pressing [Σ+], press [f] [Σ−]. This deletes the value(s) and decrements n.
2. Enter the correct value(s) using [Σ+].

If the incorrect values were the ones just entered, you can simply press [f] [LASTx] to retrieve them, then [f] [Σ−] to delete them. (The incorrect y-value was still in the Y-register, and its x-value was saved in the LASTx register.)

Example. Key in the x, y-values on the left, then make the corrections shown on the right.

Initial x, y	Corrected x, y
20, 4	20, 5
400, 6	40, 6

Keys:	Display:	Description:
[f] [CLEAR] {Σ}		Clears previous statistical data, then enters two data pairs. Display shows n, the number of data pairs entered.
4 [ENTER] 20 [Σ+]	1.0000	
6 [ENTER] 400 [Σ+]	2.0000	
[f] [LASTx]	400.0000	Brings back last x-value. Last y is still in Y-register. (Press [x≷y] twice to check y.)
[f] [Σ−]	1.0000	Deletes and replaces last data pair (400, 6 to 40, 6).
6 [ENTER] 40 [Σ+]	2.0000	

4 [ENTER] 20 ■ [Σ−]	1.0000	Deletes and replaces the first pair (20, 4 to 20, 5). Still two pairs total.
5 [ENTER] 20 [Σ+]	2.0000	

Statistical Calculations

Once you have entered your statistical data, you can use the functions in the STAT menu. Press ■ [STAT] to display the STAT menu.

The STAT Menu

Menu Label	Description
{Σ}	The summation menu: n, Σx, Σy, Σx^2, Σy^2, Σxy. See "Summation Statistics."
{$\bar{x}$,$\bar{y}$}	The mean menu: $\bar{x}$, $\bar{y}$ and weighted $\bar{x}$ ($\bar{x}w$). See "Mean and Standard Deviation."
{s}	The standard-deviation menu: s_x and s_y. See "Mean and Standard Deviation."
{L.R.}	The linear-regression menu: curve-fitting (r, m, b) and linear estimation ($\hat{x}$, $\hat{y}$). See "Linear Regression."

Mean and Standard Deviation

The Mean ($\bar{x}$, $\bar{y}$) Menu.

- Press ■ [STAT] {$\bar{x}$,$\bar{y}$} {$\bar{x}$} for the arithmetic mean (average) of the x-values.
- Press ■ [STAT] {$\bar{x}$,$\bar{y}$} {$\bar{y}$} for the arithmetic mean (average) of the y-values.

- Press ■ STAT {x̄,ȳ} {x̄w} for the weighted mean of the x-values using the y-values as weights or frequencies. The weights can be integers or non-integers.

The Standard Deviation (s) Menu. Standard deviation is a measure of how dispersed the data values are about the mean.

- Press ■ STAT {s} {sx} for the standard deviation of the x-values.*
- Press ■ STAT {s} {sy} for the standard deviation of the y-values.*

Example: Mean and Standard Deviation With One Variable. Production supervisor May Kitt wants to determine how long a certain process takes. She randomly picks ten people, observes each one as he or she carries out the process, and records the number of minutes required:

15.5 9.25 10.0
12.5 12.0 8.5

Calculate the mean and standard deviation of the times. (Treat all these data as x-values.)

Keys:	Display:	Description:
■ CLEAR {Σ}		Clears the statistics registers.
15.5 Σ+	1.0000	Enters the first time.
9.25 Σ+ 10 Σ+ 12.5 Σ+ 12 Σ+ 8.5 Σ+	3.0000 5.0000 6.0000	Enters the remaining data.
■ STAT {x̄,ȳ} {x̄}	11.2917	Calculates mean.

* This calculates the *sample standard deviation* (using $n-1$ as a divisor), which assumes the data is a sampling of a larger, complete set of data. If your data constitute the entire population of data, the *true population standard deviation* can be computed by calculating the mean of the original data, adding the mean to the statistical data using Σ+, and then calculating the standard deviation.

■ STAT {s} {sx}	2.5808	Calculates standard deviation.

Example: Weighted Mean. A manufacturing company purchases a certain part four times a year. Last year's purchases were:

Price per Part (x)	$4.25	$4.60	$4.70	$4.10
Number of Parts (y)	250	800	900	1000

Find the mean price paid for this part. Remember to enter y, the weight (frequency), before x, the price.

Keys:	**Display:**	**Description:**
■ CLEAR {Σ}		Clears the statistics registers.
250 ENTER 4.25 Σ+	1.0000	Enters the data and their weights.
800 ENTER 4.6 Σ+	2.0000	
900 ENTER 4.7 Σ+	3.0000	
1000 ENTER 4.1 Σ+	4.0000	
■ STAT {x̄,ȳ} {x̄w}	4.4314	Calculates mean price weighted for quantity purchased.

Linear Regression

Linear regression (also called *linear estimation*) is a statistical method for finding a straight line that best fits a set of x,y-data. Be sure to enter your data values before using these functions.

- To find an estimated value for x (or y), *first* key in a given hypothetical value for y (or x), *then* press ■ STAT {L.R.} {x̂} (or {ŷ}).
- To find the values that define the line that best fits your data, press ■ STAT {L.R.} followed {r}, {m}, or {b}.

The Linear Regression (L.R.) Menu

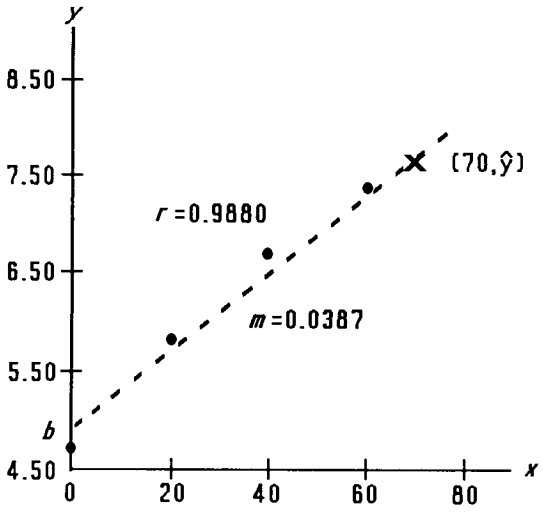
Menu Label	Description
{x̂}	Estimates (predicts) x for a given hypothetical value of y, based on the line calculated to fit the data.
{ŷ}	Estimates (predicts) y for a given hypothetical value of x, based on the line calculated to fit the data.
{r}	Correlation coefficient for the (x,y) data. The correlation coefficient is a number in the range -1 through $+1$ that measures how closely the calculated line fits the data.
{m}	Slope of the calculated line.
{b}	y-intercept of the calculated line.

Example: Curve Fitting. The yield of a new variety of rice depends on its rate of fertilization with nitrogen. For the following data, determine the linear relationship: the correlation coefficient, the slope, and the y-intercept.

X, Nitrogen Applied (kg per hectare)	0.00	20.00	40.00	60.00	80.00
Y, Grain Yield (metric tons per hectare)	4.63	5.78	6.61	7.21	7.78

Keys:	Display:	Description:
■ CLEAR {Σ}		Clears any previous statistical data.
4.63 ENTER 0 Σ+ 5.78 ENTER 20 Σ+ 6.61 ENTER 40 Σ+ 7.21 ENTER 60 Σ+ 7.78 ENTER 80 Σ+	5.0000	Enters data; displays n: 5 data pairs entered.
■ STAT {L.R.}	x̂ ŷ r m b	Displays linear-regression menu.
{r}	0.9880	Correlation coefficient; the data closely approximate a straight line.

■ STAT {L.R.} {m}	0.0387	Slope of the line.
■ STAT {L.R.} {b}	4.8560	y-intercept.



What if 70 kg of nitrogen fertilizer were applied to the rice field? Predict the grain yield based on the above statistics.

Keys:	Display:	Description:
70	70_	Enters hypothetical x-value.
■ STAT {L.R.} {ŷ}	7.5615	The predicted yield in tons per hectare.

Limitations on Precision of Data

Since the calculator uses finite precision (12 to 15 digits), it follows that there are limitations to calculations due to rounding. Here are two examples:

Normalizing Close, Large Numbers. The calculator might be unable to correctly calculate the standard deviation and linear regression for a variable whose data values differ by a relatively small amount. To avoid this, normalize the data by entering each value as the difference from one central value (such as the mean). For normalized x-values, this difference must then be added back to the calculation of $\bar{x}$ and $\hat{x}$, and $\hat{y}$ and b must also be adjusted. For example, if your x-values were 7776999, 7777000, and 7777001, you should enter the data as -1, 0, and 1; then add 7777000 back to $\bar{x}$ and $\hat{x}$. For b, add back 7777000 $\times$ m. To calculate $\hat{y}$, be sure to supply an x-value that is less 7777000.

Similar inaccuracies can result if your x and y values have greatly different magnitudes. Again, scaling the data can avoid this problem.

Effect of Deleted Data. Executing ■ [Σ−] does not delete any rounding errors that might have been generated in the statistics registers by the original data values. This difference is not serious unless the incorrect data have a magnitude that is enormous compared with the correct data; in such a case, it would be wise to clear and reenter all the data.

Summation Values and the Statistics Registers

The statistics registers are six unique locations in memory that store the accumulation of the six summation values.

Summation Statistics

Pressing ■ [STAT] {Σ} gives you access to the contents of the statistics registers:

- Press {n} to see the number of accumulated data sets.
- Press {x} to see the sum of the x-values.
- Press {y} to see the sum of the y-values.

- Press {x^2}, {y^2}, and {xy} to see the sums of the squares and the sum of the products, values that are of interest for performing other statistical calculations besides those provided by the calculator.

The Statistics Registers in Calculator Memory

The memory space (48 bytes) for the statistics registers is automatically allocated (if it doesn't exist already) when you press [Σ+] or [Σ−]. The registers are deleted and the memory deallocated when you execute ■ [CLEAR] {Σ}.

If not enough calculator memory is available to hold the statistics registers when you first press [Σ+] (or [Σ−]), the calculator displays MEMORY FULL. You will need to clear variables or programs (or both) to make room for the statistics registers before you can enter statistical data. Refer to "Managing Calculator Memory" in appendix B.

Part 4

Application Programs

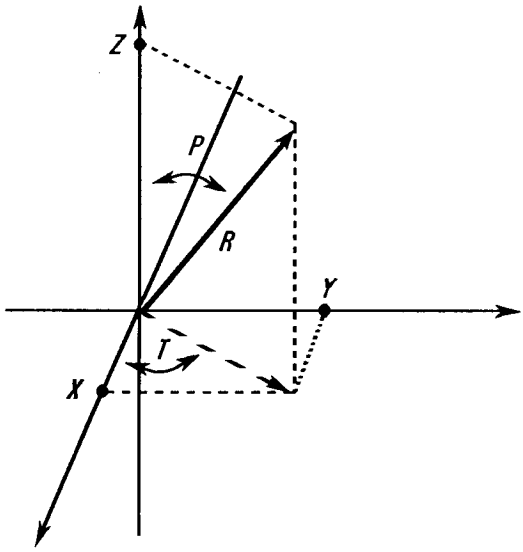
12

Mathematics Programs

The memory usage and checksum for each program label can be checked using the catalog of programs (page 85).

Vector Operations

This program performs the basic vector operations of addition, subtraction, cross product, and dot (or scalar) product. The program uses three-dimensional vectors and provides input and output in rectangular or polar form. Angles between vectors can also be found.



Vector Coordinate Systems

This program uses the following equations.

Coordinate conversion:

$$X = R \sin(P) \cos(T) \qquad R = \sqrt{X^2 + Y^2 + Z^2}$$

$$Y = R \sin(P) \sin(T) \qquad T = \arctan \frac{Y}{X}$$

$$Z = R \cos(P) \qquad P = \arctan \frac{Z}{\sqrt{X^2 + Y^2}}$$

Vector addition and subtraction:

$$\mathbf{v}_1 + \mathbf{v}_2 = (X + U)\mathbf{i} + (Y + V)\mathbf{j} + (Z + W)\mathbf{k}$$

$$\mathbf{v}_2 - \mathbf{v}_1 = (U - X)\mathbf{i} + (V - Y)\mathbf{j} + (W - Z)\mathbf{k}$$

Cross product:

$$\mathbf{v}_1 \times \mathbf{v}_2 = (YW - ZV)\mathbf{i} + (ZU - XW)\mathbf{j} + (XV - YU)\mathbf{k}$$

Dot product:

$$D = XU + YV + ZW$$

Angle between vectors (γ):

$$G = \arccos \frac{D}{R_1 \times R_2},$$

where

$$\mathbf{v}_1 = X\mathbf{i} + Y\mathbf{j} + Z\mathbf{k}$$

and

$$\mathbf{v}_2 = U\mathbf{i} + V\mathbf{j} + W\mathbf{k}$$

The vector displayed by the input routines (LBL P and LBL R) is V_1.

Program Listing:

Program Lines:	Description:
R01 LBL R	Defines the beginning of the rectangular input/display routine.
R02 INPUT X	Displays or accepts input of X.
R03 INPUT Y	Displays or accepts input of Y.
R04 INPUT Z	Displays or accepts input of Z.
Bytes and Checksum: 006.0, 80FB	
Q01 LBL Q	Defines beginning of rectangular-to-polar conversion process.
Q02 RCL Y	
Q03 RCL X	
Q04 y,x→θ,r	Calculates $\sqrt{(X^2 + Y^2)}$ and arctan(Y/X).
Q05 x<>y	
Q06 STO T	Saves T = arctan(Y/X).
Q07 R↓	Gets $\sqrt{(X^2 + Y^2)}$ back.
Q08 RCL Z	
Q09 y,x→θ,r	Calculates $\sqrt{(X^2 + Y^2 + Z^2)}$ and P.
Q10 STO R	Saves R.
Q11 x<>y	
Q12 STO P	Saves P.
Bytes and Checksum: 018.0, D6D5	
P01 LBL P	Defines the beginning of the polar input/display routine.
P02 INPUT R	Displays or accepts input of R.
P03 INPUT T	Displays or accepts input of T.
P04 INPUT P	Displays or accepts input of P.
P05 RCL T	
P06 RCL P	
P07 RCL R	
P08 θ,r→y,x	Calculates R cos(P) and Rsin(P).
P09 STO Z	Stores $Z = R$ cos(P).
P10 R↓	
P11 θ,r→y,x	Calculates R sin(P) cos(T) and R sin(P) sin(T).
P12 STO X	Saves $X = R$ sin(P) cos(T).
P13 x<>y	

```
P14 STO Y
P15 GTO P
```
Saves $Y = R \sin(P) \sin(T)$.
Loops back for another display of polar form.

Bytes and Checksum: 022.5, AA98

```
E01 LBL E
E02 RCL X
E03 STO U
E04 RCL Y
E05 STO V
E06 RCL Z
E07 STO W
E08 GTO Q
```
Defines the beginning of the vector-enter routine.
Copies values in *X*, *Y* and *Z* to *U*, *V* and *W* respectively.
Loops back for polar conversion and display/input.

Bytes and Checksum: 012.0, 7137

```
X01 LBL X
X02 RCL X
X03 RCL U
X04 STO X
X05 x<>y
X06 STO U
X07 RCL Y
X08 RCL V
X09 STO Y
X10 x<>y
X11 STO V
X12 RCL Z
X13 RCL W
X14 STO Z
X15 x<>y
X16 STO W
X17 GTO Q
```
Defines beginning of vector-exchange routine.
Exchanges *X*, *Y* and *Z* with *U*, *V* and *W* respectively.
Loops back for polar conversion and display/input.

Bytes and Checksum: 025.5, EAD8

```
A01 LBL A
A02 RCL X
A03 RCL+ U
A04 STO X
A05 RCL V
A06 RCL+ Y
A07 STO Y
A08 RCL Z
A09 RCL+ W
A10 STO Z
A11 GTO Q
```

Defines beginning of vector-addition routine.

Saves $X + U$ in X.

Saves $V + Y$ in Y.

Saves $Z + W$ in Z.

Loops back for polar conversion and display/input.

Bytes and Checksum: 016.5, F888

```
S01 LBL S
S02 -1
S03 STO× X
S04 STO× Y
S05 STO× Z
S06 GTO A
```

Defines the beginning of the vector-subtraction routine.

Multiplies X, Y and Z by (-1) to change the sign.

Goes to the vector-addition routine.

Bytes and Checksum: 017.0, 250B

```
C01 LBL C
C02 RCL Y
C03 RCL× W
C04 RCL Z
C05 RCL× V
C06 -
C07 RCL Z
C08 RCL× U
C09 RCL X
C10 RCL× W
C11 -
```

Defines the beginning of the cross-product routine.

Calculates $(YW - ZV)$, which is the X component.

Calculates $(ZU - WX)$, which is the Y component.

```
C12 RCL X
C13 RCL× V
C14 RCL Y
C15 RCL× U
C16 -
C17 STO Z
```
Stores ($XV - YU$), which is the Z component.

```
C18 R↓
C19 STO Y
```
Stores Y component.

```
C20 R↓
C21 STO X
```
Stores X component.

```
C22 GTO Q
```
Loops back for polar conversion and display/input.

Bytes and Checksum: 033.0, D74B

```
D01 LBL D
```
Defines beginning of dot-product and vector-angle routine.

```
D02 RCL X
D03 RCL× U
D04 RCL Y
D05 RCL× V
D06 +
D07 RCL Z
D08 RCL× W
D09 +
D10 STO D
```
Stores the dot product of $XU + YV + ZW$.

```
D11 VIEW D
```
Displays the dot product.

```
D12 RCL D
D13 RCL R
D14 ÷
```
Divides the dot product by the magnitude of the X-, Y-, Z-vector.

```
D15 RCL W
D16 RCL V
D17 RCL U
D18 y,x→θ,r
D19 x<>y
D20 R↓
D21 y,x→θ,r
```
Calculates the magnitude of the U, V, W vector.

```
D22 x<>y
D23 R↓
```

```
D24 ÷          Divides previous result by the magnitude.
D25 ACOS       Calculates angle.
D26 STO G
D27 VIEW G     Displays angle.
D28 GTO P      Loops back for polar display/input.
```
Bytes and Checksum: 042.0, 739F

Flags Used: None.

Memory Required: 280.5 bytes: 192.5 for program, 88 for variables.

Remarks: The length of routine S can be shortened by 6.5 bytes. The value −1 as shown uses 9.5 bytes. If it appears as 1 followed by +/− , it will require only 3 bytes. To do this, you must key in a dummy step between the 1 and the +/−, and then delete the dummy step.

The terms "polar" and "rectangular," which refer to two-dimensional systems, are used instead of the proper three-dimensional terms of "spherical" and "Cartesian." This stretch of terminology allows the labels to be associated with their function without confusing conflicts. For instance, if LBL C had been associated with Cartesian coordinate input, it would not have been available for cross product.

Program Instructions:

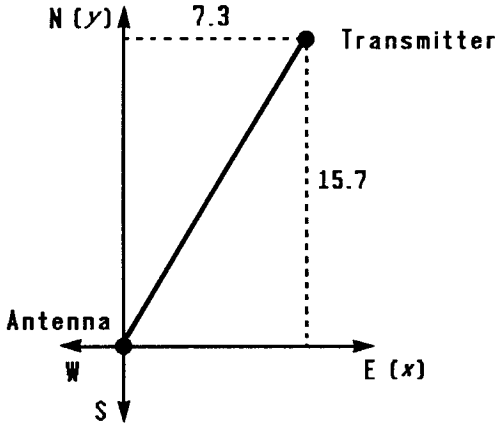
1. Key in the program routines; press [C] when done.
2. If your vector is in rectangular form, press [XEQ] R and go to step 4. If your vector is in polar form, press [XEQ] P and continue with step 3.
3. Key in *R* and press [R/S], key in *T* and press [R/S], and key in *P* and press [R/S]. Continue at step 5.
4. Key in *X* and press [R/S], key in *Y* and press [R/S], and key in *Z* and press [R/S].
5. To key in a second vector, press [XEQ] E (for enter) and go to step 2.

6. Perform desired vector operation:
   - **a.** Add vectors by pressing [XEQ] A;
   - **b.** Subtract vector one from vector two by pressing [XEQ] S;
   - **c.** Compute the cross product by pressing [XEQ] C;
   - **d.** Compute the dot product by pressing [XEQ] D and the angle between vectors by pressing [R/S].
7. Optional: to review $\mathbf{v}_1$ in polar form, press [XEQ] P, then press [R/S] repeatedly to see the individual elements.
8. Optional: to review $\mathbf{v}_1$ in rectangular form, press [XEQ] R, then press [R/S] repeatedly to see the individual elements.
9. If you added, subtracted, or computed the cross product, $\mathbf{v}_1$ has been replaced by the result. $\mathbf{v}_2$ is not altered. To continue calculations based on the result, remember to press [XEQ] E before keying in a new vector.
10. Go to step 2 to continue vector calculations.

Variables Used:

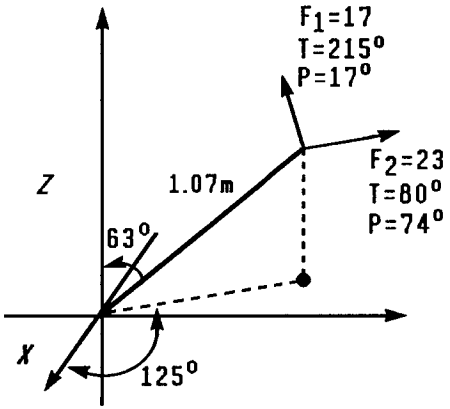
X, Y, Z	The rectangular components of $\mathbf{v}_1$.
U, V, W	The rectangular components of $\mathbf{v}_2$.
R, T, P	The radius, the angle in the x-y plane (θ), and the angle from the Z axis of $\mathbf{v}_1$ (Φ).
D	The dot product.
G	The angle between vectors (γ).

Example 1. A microwave antenna is to be pointed at a transmitter which is 15.7 kilometers North, 7.3 kilometers East and 0.76 kilometers below. Use the rectangular to polar conversion capability to find the total distance and the direction to the transmitter.



Keys:	Display:	Description:
[XEQ] R	X?*value*	Starts rectangular input/display routine.
7.3 [R/S]	Y?*value*	Sets X equal to 7.3.
15.7 [R/S]	Z?*value*	Sets Y equal to 15.7.
.76 [+/-] [R/S]	R=17.3308	Sets Z equal to -0.76 and calculates R, the radius.
[R/S]	T=65.0631	Calculates T, the angle in the x/y plane.
[R/S]	P=92.5134	Calculates P, the angle from the z-axis.

Example 2. What is the moment at the origin of the lever shown below? What is the component of force along the lever? What is the angle between the resultant of the force vectors and the lever?



First, add the force vectors.

Keys:	Display:	Description:
[XEQ] P	R?*value*	Starts polar input routine.
17 [R/S]	T?*value*	Sets radius equal to 17.
215 [R/S]	P?*value*	Sets T equal to 215.
17 [R/S]	R?17.0000	Sets P equal to 17.
[XEQ] E	R?17.0000	Enters vector by copying it into $\mathbf{v}_2$.
23 [R/S]	T?-145.0000	Sets radius of $\mathbf{v}_1$ equal to 23.
80 [R/S]	P?17.0000	Sets T equal to 80.
74 [R/S]	R?23.0000	Sets P equal to 74.
[XEQ] A	R?29.4741	Adds the vectors and displays the resultant R.
[R/S]	T?90.7032	Displays T of resultant vector.

R/S	P?39.9445	Displays P of resultant vector.
XEQ E	R?29.4741	Enters resultant vector.

Since the moment equals the cross product of the radius vector and the force vector ($\mathbf{r} \times \mathbf{F}$), key in the vector representing the lever and take the cross product.

Keys:	**Display:**	**Description:**
1.07 R/S	T?90.7032	Sets R equal to 1.07.
125 R/S	P?39.9445	Sets T equal to 125.
63 R/S	R?1.0700	Sets P equal to 63.
XEQ C	R?18.0209	Calculates cross product and displays R of result.
R/S	T?55.3719	Displays T of cross product.
R/S	P?124.3412	Displays P of cross product.
XEQ R	X?8.4554	Displays rectangular form of cross product.
R/S	Y?12.2439	
R/S	Z?-10.1660	

The dot product can be used to resolve the force (still in $\mathbf{v}_2$) along the axis of the lever.

Keys:	Display:	Description:
XEQ P	R?18.0209	Starts polar input routine.
1 R/S	T?55.3719	Defines the radius as one unit vector.
125 R/S	P?124.3412	Sets T equal to 125.
63 R/S	R?1.0000	Sets P equal to 63.
XEQ D	D=24.1882	Calculates dot product.
R/S	G=34.8490	Calculates angle between resultant force vector and lever.
R/S	R?1.0000	Gets back to input routine.

Solutions of Simultaneous Equations—Determinant Method

This program solves simultaneous linear equations in two or three unknowns. The program uses Cramer's method, also know as the method of determinants.

Given a system of three linear equations

$$AX + DY + GZ = J$$

$$BX + EY + HZ = K$$

$$CX + FY + IZ = L$$

the three unknowns X, Y, and Z may be computed from determinants.

$$X = \frac{Det_x}{Det} \qquad Y = \frac{Det_y}{Det} \qquad Z = \frac{Det_z}{Det}$$

$$Det = \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix} \qquad Det_x = \begin{bmatrix} J & D & G \\ K & E & H \\ L & F & I \end{bmatrix}$$

$$Det_y = \begin{bmatrix} A & J & G \\ B & K & H \\ C & L & I \end{bmatrix} \qquad Det_z = \begin{bmatrix} A & D & J \\ B & E & K \\ C & F & L \end{bmatrix}$$

Program Listing:

Program Lines:	**Description:**
A01 LBL A	Starting point for input of all known values.
A02 1.012	Loop-control value: loops from 1 to 12, 1 at a time.
A03 STO i	Stores control value in index variable.
Bytes and Checksum: 012.5, 7878	
L01 LBL L	Starts the input loop.
L02 INPUT(i)	Prompts for and stores the variable pointed to by *i*.
L03 ISG i	Adds one to *i*.
L04 GTO L	If *i* is less than 13, goes back to LBL L and gets the next value.
L05 GTO A	Returns to LBL A to review values.
Bytes and Checksum: 007.5, C1DE	
S01 LBL S	Starting point for simultaneous equation solutions.
S02 9	Index value of *I* for indirect addressing.
S03 STO i	Stores index value.

```
S04 XEQ E          Exchanges solution column and coefficients
                   column containing I.
S05 XEQ D          Calculates determinant.
S06 STO Z          Saves determinant in Z.
S07 XEQ E          Restores determinant to original form.
S08 6              Index value of F for indirect addressing.
S09 STO i          Stores index value.
S10 XEQ E          Exchanges solution column and column
                   containing F.
S11 XEQ D          Calculates determinant.
S12 STO Y          Saves determinant in Y.
S13 XEQ E          Restores determinant to original form.
S14 3              Index value of C for indirect addressing.
S15 STO i          Stores index value in index variable.
S16 XEQ E          Exchanges solution column and column
                   containing F.
S17 XEQ D          Calculates determinant.
S18 STO X          Saves determinant in X.
S19 XEQ E          Restores determinant to original form.
S20 XEQ D          Calculates determinant of original
                   coefficients.
S21 STO÷ X         Divides by original determinant.
S22 STO÷ Y
S23 STO÷ Z
S24 RCL X          Recalls and displays results for X, Y and Z.
S25 VIEW X
S26 RCL Y
S27 VIEW Y
S28 RCL Z
S29 VIEW Z
S30 RTN            Returns to the calling program or to
                   PRGM TOP.
```

Bytes and Checksum: 045.0, 3971

```
E01 LBL E          This routine exchanges columns for
                   Cramer's rule.
E02 RCL(i)         Gets last element from column of coefficient
                   determinant.
E03 RCL L          Gets last element from solution vector.
E04 STO(i)         Saves vector element in determinant.
```

E05 x<>y	Gets the coefficient element back.
E06 STO L	Saves the element in the vector.
E07 DSE i	Sets index value to point to middle element in column of determinant.
E08 RCL(i)	Gets middle element from column of determinant.
E09 RCL K	Gets middle element from vector.
E10 STO(i)	Saves vector element in determinant.
E11 x<>y	Gets the coefficient element back.
E12 STO K	Saves the coefficient element in the vector.
E13 DSE i	Sets index value to point to top element in column of determinant.
E14 RCL(i)	Gets top element from column of determinant.
E15 RCL J	Gets top element from vector.
E16 STO(i)	Saves vector element in determinant.
E17 x<>y	Gets the determinant element back.
E18 STO J	Saves the determinant element in the vector.
E19 2	
E20 STO+ i	Restores *i* to its original value when routine started.
E21 RTN	Returns to the calling program or to PRGM TOP.

Bytes and Checksum: 031.5, 8420

D01 LBL D	This routine calculates the determinant.
D02 RCL A	
D03 RCL× E	
D04 RCL× I	Calculates $A \times E \times I$.
D05 RCL D	
D06 RCL× H	
D07 RCL× C	
D08 +	Calculates $(A \times E \times I) + (D \times H \times C)$.
D09 RCL G	
D10 RCL× F	
D11 RCL× B	
D12 +	Calculates $(A \times E \times I) + (D \times H \times C) + (G \times F \times B)$.
D13 RCL G	
D14 RCL× E	

D15 RCL× C	
D16 -	$(A \times E \times I) + (D \times H \times C) + (G \times F \times B) - (G \times E \times C)$.
D17 RCL A	
D18 RCL× F	
D19 RCL× H	
D20 -	$(A \times E \times I) + (D \times H \times C) + (G \times F \times B) - (G \times E \times C) - (A \times F \times H)$.
D21 RCL D	
D22 RCL× B	
D23 RCL× I	
D24 -	$(A \times E \times I) + (D \times H \times C) + (G \times F \times B) - (G \times E \times C) - (A \times F \times H) - (D \times B \times I)$.
D25 RTN	Returns to the calling program or to PRGM TOP.

Bytes and Checksum: 037.5, 152E

Flags Used: None.

Memory Required: 262 bytes: 134 for program, 128 for variables.

Program Instructions:

1. Key in the program routines; press [C] when done.
2. Press [XEQ] A to input coefficients (that is, *A* through *L*) of linear equations.
3. Key in coefficient (*A* through *L*) at each prompt and press [R/S].
4. Optional: to compute determinant of a 3 × 3 system, [XEQ] D.
5. Compute solution to system of equations by pressing [XEQ] S.
6. See value of *X* and press [R/S] to see the value of *Y*.
7. Press [R/S] to see the value of *Z*.
8. For a new case, go back to step 2.

Variables Used:

A through *I*	Coefficients of equations.
J through *L*	Right-hand sides of equations.
X through *Z*	Unknowns.
i	Loop-control value (index variable).

Remarks: This program is for a system of two or three equations (that is, a matrix of $n \leq 3$).

For 2 × 2 solutions use zero for coefficients *C*, *F*, *G*, *H*, and *L*. Use 1 for coefficient *I*. For non-square matrices, use zero for the "missing" coefficients.

Not all systems of equations have solutions. If not, they cause the error `DIVIDE BY 0` at line S21.

Example 1. For the system below, compute the determinant and the system solution. Then substitute the values back into the first equation to verify that the left side of the equation is actually equal to the right side (1).

$$23X + 15Y + 17Z = 1$$

$$8X + 11Y - 6Z = 1$$

$$4X + 15Y + 12Z = 1$$

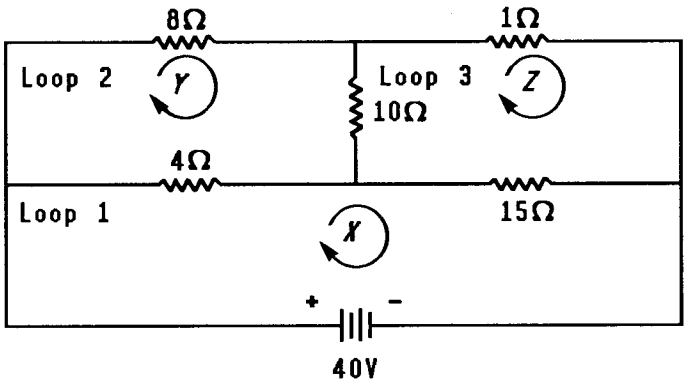
Keys:	**Display:**	**Description:**
[XEQ] A	A?*value*	Starts input routine.
23 [R/S]	B?*value*	Sets first coefficient, *A*, equal to 23.
8 [R/S]	C?*value*	Sets *B* equal to 8.
4 [R/S]	D?*value*	Sets *C* equal to 4.
15 [R/S]	E?*value*	Sets *D* equal to 15.
⋮	⋮	Continues entry for all values (*E* through *L*).
1 [R/S]	A?23.0000	Returns to first coefficient entered.
[XEQ] D	4,598.0000	Calculates the determinant.

XEQ S	X=0.0043	Solves system of equations and displays X.
R/S	Y=0.0787	Displays Y.
R/S	Z=-0.0165	Displays Z.

Now, to verify the result:

Keys:	Display:	Description:
23 RCL × X	0.1000	Multiplies X by 23.
15 RCL × Y	1.1809	Multiplies Y by 15.
+	1.2810	Adds the last two results.
17 RCL × Z	-0.2810	Multiplies Z by 17.
+	1.0000	Completes the left side of the equation. Since the left and right sides are both equal to one (to 11 significant digits), the solution is correct.

Example 2. Solve for the loop currents in the circuit below:



First write the equations for the voltage drops around each loop.

For loop 1: $4X - 4Y + 15X - 15Z - 40 = 0$

For loop 2: $4Y - 4X + 8Y + 10Y - 10Z = 0$

For loop 3: $10Z - 10Y + Z + 15Z - 15X = 0$

Combining like terms within each equation produces

$$19X - 4Y - 15Z = 40$$

$$-4X + 22Y - 10Z = 0$$

$$-15X - 10Y + 26Z = 0$$

Keys:	**Display:**	**Description:**
[XEQ] A	A?*value*	Starts input routine.
19 [R/S]	B?*value*	Sets first coefficient, *A*, equal to 19.
4 [+/-] [R/S]	C?*value*	Sets *B* equal to −4.
15 [+/-] [R/S]	D?*value*	Sets *C* equal to −15.
⋮		Continues entry for *D* through *L*.
0 [R/S]	A?19.0000	Enters *L* and returns to first coefficient entered.
[XEQ] S	X=7.8601	Solves system of equations and displays *X*.
[R/S]	Y=4.2298	Displays *Y*.
[R/S]	Z=6.1615	Displays *Z*.

Solutions of Simultaneous Equations—Matrix Inversion Method

This program solves simultaneous linear equations in two or three unknowns. It does this through matrix inversion and matrix multiplication.

A system of three linear equations

$$AX + DY + GZ = J$$

$$BX + EY + HZ = K$$

$$CX + FY + IZ = L$$

can be represented by the matrix equation below.

$$\begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} J \\ K \\ L \end{bmatrix}$$

The matrix equation may be solved for X, Y, and Z by multiplying the result matrix by the inverse of the coefficient matrix.

$$\begin{bmatrix} A' & D' & G' \\ B' & E' & H' \\ C' & F' & I' \end{bmatrix} \begin{bmatrix} J \\ K \\ L \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Specifics regarding the inversion process are given in the comments for the inversion routine, I.

Program Listing:

Program Lines:	Description:
A01 LBL A	Starting point for input of coefficients.
A02 1.012	Loop-control value: loops from 1 to 12, 1 at a time.
A03 STO i	Stores control value in index variable.
Bytes and Checksum: 012.5, 7878	
L01 LBL L	Starts the input loop.
L02 INPUT(i)	Prompts for and stores the variable addressed by *i*.
L03 ISG i	Adds one to *i*.
L04 GTO L	If *i* is less than 13, goes back to LBL L and gets the next value.
L05 GTO A	Returns to LBL A to review values.
Bytes and Checksum: 007.5, C1DE	
I01 LBL I	This routine inverts a 3 × 3 matrix.
I02 XEQ D	Calculates determinant and saves value for the division loop, J.
I03 STO W	
I04 RCL A	
I05 RCL× I	
I06 RCL C	
I07 RCL× G	
I08 -	
I09 STO X	Calculates E' × determinant = $AI - CG$.
I10 RCL C	
I11 RCL× D	
I12 RCL A	
I13 RCL× F	
I14 -	
I15 STO Y	Calculates F' × determinant = $CD - AF$.
I16 RCL B	
I17 RCL× G	
I18 RCL A	
I19 RCL× H	
I20 -	
I21 STO Z	Calculates H' × determinant = $BG - AH$.
I22 RCL A	

```
I23 RCL× E
I24 RCL B
I25 RCL× D
I26 -
I27 STO i          Calculates I′ × determinant = AE − BD.
I28 RCL E
I29 RCL× I
I30 RCL F
I31 RCL× H
I32 -
I33 STO A          Calculates A′ × determinant = EI − FH.
I34 RCL C
I35 RCL× H
I36 RCL B
I37 RCL× I
I38 -              Calculates B′ × determinant = CH − BI.
I39 RCL B
I40 RCL× F
I41 RCL C
I42 RCL× E
I43 -
I44 STO C          Calculates C′ × determinant = BF − CE.
I45 R↓
I46 STO B          Stores B′.
I47 RCL F
I48 RCL× G
I49 RCL D
I50 RCL× I
I51 -              Calculates D′ × determinant = FG − DI.
I52 RCL D
I53 RCL× H
I54 RCL E
I55 RCL× G
I56 -
I57 STO G          Calculates G′ × determinant = DE − EG.
I58 R↓
I59 STO D          Stores D′.
I60 RCL i
I61 STO I          Stores I′.
I62 RCL X
I63 STO E          Stores E′.
```

```
I64 RCL Y
I65 STO F          Stores F'.
I66 RCL Z
I67 STO H          Stores H'.
I68 9
I69 STO i          Sets index value to point to last element of
                   matrix.
I70 RCL W          Recalls value of determinant.
```
Bytes and Checksum: 105.0, E5C1

```
J01 LBL J          This routine completes inverse by dividing
                   by determinant.
J02 STO÷(i)        Divides element.
J03 DSE i          Decrements index value so that it points
                   closer to A.
J04 GTO J          Loops for next value.
J05 RTN            Returns to the calling program or to
                   PRGM TOP.
```
Bytes and Checksum: 007.5, A354

```
M01 LBL M          This routine multiplies a column matrix and
                   a 3 × 3 matrix.
M02 7              Sets index value to point to last element in
                   first row.
M03 XEQ N
M04 8              Sets index value to point to last element in
                   second row.
M05 XEQ N
M06 9              Sets index value to point to last element in
                   third row.
```
Bytes and Checksum: 009.0, 0A85

```
N01 LBL N          This routine calculates product of column
                   vector and row pointed to by index value.
N02 STO i          Saves index value in i.
N03 RCL J          Recalls J from column matrix.
N04 RCL K          Recalls K from column matrix.
N05 RCL L          Recalls L from column vector.
N06 RCL×(i)        Multiplies by last element in row.
N07 XEQ P          Multiplies by second element in row and
                   adds.
```

```
N08 XEQ P
```
Multiplies by third element in row and adds.

```
N09 23
```
Sets index value to display X, Y, or Z based on input row.

```
N10 STO+ i
N11 R↓
```
Gets result back.

```
N12 STO(i)
```
Stores result.

```
N13 VIEW(i)
```
Displays result.

```
N14 RTN
```
Returns to the calling program or to PRGM TOP.

Bytes and Checksum: 021.0, BBBF

```
P01 LBL P
```
This routine multiplies and adds values within a row.

```
P02 x<>y
```
Gets next column value.

```
P03 DSE i
```
Sets index value to point to next row value.

```
P04 DSE i
P05 DSE i
P06 RCL×(i)
```
Multiplies column value by row value.

```
P07 +
```
Adds product to previous sum.

```
P08 RTN
```
Returns to the calling program.

Bytes and Checksum: 012.0, 520E

```
D01 LBL D
```
This routine calculates the determinant.

```
D02 RCL A
D03 RCL× E
D04 RCL× I
```
Calculates $A \times E \times I$.

```
D05 RCL D
D06 RCL× H
D07 RCL× C
D08 +
```
Calculates $(A \times E \times I) + (D \times H \times C)$.

```
D09 RCL G
D10 RCL× F
D11 RCL× B
D12 +
```
Calculates $(A \times E \times I) + (D \times H \times C) + (G \times F \times B)$.

```
D13 RCL G
D14 RCL× E
D15 RCL× C
D16 -
```
$(A \times E \times I) + (D \times H \times C) + (G \times F \times B) - (G \times E \times C)$.

```
D17 RCL A
D18 RCL× F
D19 RCL× H
D20 -                (A×E×I) + (D×H×C) + (G×F×B) −
                     (G×E×C) − (A×F×H).
D21 RCL D
D22 RCL× B
D23 RCL× I
D24 -                (A×E×I) + (D×H×C) + (G×F×B) −
                     (G×E×C) − (A×F×H) − (D×B×I).
D25 RTN              Returns to the calling program or to
                     PRGM TOP.
```

Bytes and Checksum: 037.5, 152E

Flags Used: None.

Memory Required: 348 bytes: 212 for program, 136 for variables.

Program Instructions:

1. Key in the program routines; press [C] when done.
2. Press [XEQ] A to input coefficients of matrix and column vector.
3. Key in coefficient or vector value (*A* through *L*) at each prompt and press [R/S].
4. Optional: press [XEQ] D to compute determinant of 3 × 3 system.
5. Press [XEQ] I to compute inverse of 3 × 3 matrix.
6. Optional: press [XEQ] A and repeatedly press [R/S] to review the values of the inverted matrix.
7. Press [XEQ] M to multiply the inverted matrix by the column vector and to see the value of *X*. Press [R/S] to see the value of *Y*, then press [R/S] again to see the value of *Z*.
8. For a new case, go back to step 2.

Variables Used:

A through *I*	Coefficients of matrix.
J through *L*	Column vector values.
W	Scratch variable used to store the determinant.
X through *Z*	Output vector values; also used for scratch.
i	Loop-control value (index variable); also used for scratch.

Remarks: For 2 × 2 solutions use zero for coefficients *C*, *F*, *H*, *G* and for *L*. Use 1 for coefficient *I*.

Not all systems of equations have solutions.

Note that routines A, L, and D are common to this program and to the "Solutions of Simultaneous Equations—Determinant Method" program.

Example. For the system below, compute the inverse and the system solution. Review the inverted matrix. Invert the matrix again and review the result to make sure that the original matrix is returned.

$$23X + 15Y + 17Z = 31$$

$$8X + 11Y - 6Z = 17$$

$$4X + 15Y + 12Z = 14$$

Keys:	Display:	Description:
XEQ A	A?*value*	Starts input routine.
23 R/S	B?*value*	Sets first coefficient, *A*, equal to 23.
8 R/S	C?*value*	Sets *B* equal to 8.
4 R/S	D?*value*	Sets *C* equal to 4.
15 R/S	E?*value*	Sets *D* equal to 15.
⋮	⋮	Continues entry for *D* through *L*.

14 [R/S]	A?23.0000	Returns to first coefficient entered.
[XEQ] I	4,598.0000	Calculates the inverse and displays the determinant.
[XEQ] M	X=0.9306	Multiplies by column vector to compute X.
[R/S]	Y=0.7943	Calculates and displays Y.
[R/S]	Z=-0.1364	Calculates and displays Z.
[XEQ] A	A?0.0483	Begins review of the inverted matrix.
[R/S]	B?-0.0261	Displays next value.
[R/S]	C?0.0165	Displays next value.
[R/S]	D?0.0163	Displays next value.
[R/S]	E?0.0452	Displays next value.
[R/S]	F?-0.0620	Displays next value.
[R/S]	G?-0.0602	Displays next value.
[R/S]	H?0.0596	Displays next value.
[R/S]	I?0.0289	Displays next value.
[XEQ] I	0.0002	Inverts inverse to produce original matrix.
[XEQ] A	A?23.0000	Begins review of inverted inverted matrix.
[R/S]	B?8.0000	Displays next value,...
⋮	⋮	...and so on.

Quadratic Equation

This program uses the quadratic formula to solve for the real and complex roots of a second-degree polynomial.

A polynomial of degree two

$$ax^2 + bx + c = 0$$

can be solved for x using the quadratic formula

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

where $b^2 - 4ac$ is the discriminant. In the case of complex roots (where the discriminant is negative), the real part is

$$R = \frac{-b}{2a},$$

while the imaginary part is

$$I = \pm \frac{i\sqrt{|b^2 - 4ac|}}{2a}.$$

For real roots, the program always calculates the root of the greatest absolute value first. It does this to minimize inaccuracies that can be introduced if the square root of the discriminant is nearly equal to b. Once the first root, x_1, is found, the second root, x_2, is computed using the relationship

$$x_2 = \frac{c}{ax_1}.$$

Numerical errors, like the one avoided by this program, are common in computer software. Any computer that uses a finite number of digits for computation will fail numerically unless care is taken in selection and implementation of the method of solution. Inaccurate

results produced by the computer are often preventable by careful software design. Example 4 illustrates the numerical problem that is avoided by this quadratic-formula program.

Program Listing:

Program Lines:	**Description:**
Q01 LBL Q	Defines the beginning of the quadratic-equation routine.
Q02 INPUT A	Prompts for and stores the value of A.
Q03 x=0?	If A is zero, goes back and asks for A again.
Q04 GTO Q	
Q05 INPUT B	Prompts for and stores the value of B.
Q06 INPUT C	Prompts for and stores the value of C.
Q07 x=0?	If C is zero, goes back and asks for all inputs again.
Q08 GTO Q	
Q09 RCL B	Recalls B.
Q10 +/-	$-B$.
Q11 CF 0	Clears flag 0. (Assumes that $(-B)$ is positive.)
Q12 x<0?	Is $(-B)$ negative?
Q13 SF 0	Sets flag 0 if it is.
Q14 RCL B	
Q15 x^2	Calculates B^2.
Q16 4	
Q17 RCL× A	
Q18 RCL× C	
Q19 -	Calculates $B^2 - 4AC$.
Q20 x<0?	Tests to see if the roots are imaginary.
Q21 GTO I	Branches to imaginary routine if they are.
Q22 SQRT	$\sqrt{(B^2 - 4AC)}$.
Q23 FS? 0	Tests to see if $(-B)$ is negative.
Q24 +/-	Selects root of largest absolute value.
Q25 +	$-B - \sqrt{(B^2 - 4AC)}$ or $-B + \sqrt{(B^2 - 4AC)}$.
Q26 2	
Q27 ÷	
Q28 RCL÷ A	Calculates X of largest absolute value.
Q29 STO X	Stores value of X with largest absolute value.

```
Q30 VIEW X          Displays X.
Q31 RCL C           Calculates second value of X.
Q32 RCL÷ A
Q33 RCL÷ X          Calculates X = C ÷ AX.
Q34 STO X           Stores second value of X.
Q35 VIEW X          Displays X.
Q36 GTO Q           Goes back for a new case.
```
Bytes and Checksum: 054.0, A04D

```
I01 LBL I           Defines the beginning of the imaginary
                    computation routine.
I02 ABS
I03 SQRT
I04 2
I05 RCL× A
I06 ÷               Calculates absolute value of
                    √(B² − 4AC) ÷ 2A.
I07 STO I           Stores the imaginary part.
I08 RCL B
I09 +/-
I10 LAST×           Retrieves 2A.
I11 ÷
I12 STO R           Stores the real part in R.
I13 RCL I           Retrieves the imaginary part of X.
I14 RCL R           Retrieves the real part of X.
I15 VIEW R          Displays the real part.
I16 VIEW I          Displays the imaginary part.
I17 GTO Q           Goes back for a new case.
```
Bytes and Checksum: 025.5, DA3B

Flags Used: Flag 0 is used to remember the sign of $(-B)$. If $(-B)$ is negative, then flag 0 is set. Flag 0 is tested later in the program to assure that the first real root computed is the one of largest absolute value. If $(-B)$ is negative (flag 0 is set), then the routine subtracts the square root of the discriminant from $(-B)$. If $(-B)$ is positive (flag 0 is clear), then the routine adds the square root.

Memory Required: 127.5 bytes: 79.5 for program, 48 for variables.

Remarks: Expanding this program to handle cubic equations would be quite easy. Since a cubic equation always has at least one real root, the SOLVE function could be used to find the root. Then synthetic division could reduce the cubic equation to a quadratic equation which would be solved by this program.

Program Instructions:

1. Key in the program routines; press [C] when done.
2. Press [XEQ] Q to start the quadratic equation routine.
3. Key in *A* and press [R/S].
4. Key in *B* and press [R/S].
5. Key in *C* and press [R/S].
6. See the first value of X, if the roots are real, or see the real part, *R*, if the roots are imaginary.
7. Press [R/S] to see the second value of X, or to see the imaginary part, *I*, if the roots are imaginary.
8. For a new case, press [R/S] and go back to step 3.

Variables Used:

A Coefficient of x^2.

B Coefficient of x.

C Constant.

X The first or second real value of x.

R The real portion of the complex root.

I The positive, imaginary part of the complex root.

Example 1. Find the roots of $3x^2 + 5x - 3 = 0$.

Keys:	Display:	Description:
XEQ Q	A?*value*	Starts the quadratic equation program.
3 R/S	B?*value*	Stores 3 in *A*.
5 R/S	C?*value*	Stores 5 in *B*.
3 +/- R/S	X=-2.1350	Stores −3 in *C* and calculates the first value of *X*.
R/S	X=0.4684	Calculates second value of *X*.

Example 2. Find the roots of $3x^2 + 5x + 3 = 0$. Note that the only difference between this problem and example 1 is the sign of *C*. If you have already run example 1, all you have to do is change the sign of *C*:

Keys:	Display:	Description:
R/S	A?3.0000	Resumes the program.
R/S	B?5.0000	Keeps *A*.
R/S	C?-3.0000	Keeps *B*.
+/- R/S	R=-0.8333	Changes the sign of *C* and calculates the real part of the complex root.
R/S	I=0.5528	Calculates the positive value of the imaginary root.

Example 3. A ball is thrown straight up at a velocity of 20 meters per second from a height of 2 meters. Neglecting air resistance, at what time will it reach the ground? The acceleration of gravity is 9.81 meters per second^2.

According to Newtonian mechanics, this problem may be expressed as a second degree polynomial, where T is time in seconds.

$$f(T) = (-9.81 \div 2)T^2 + 20T + 2$$

Keys:	Display:	Description:
XEQ Q	A?*value*	Starts the quadratic equation program.
9.81 +/- ENTER 2 ÷ R/S	B?*value*	Stores (−9.81/2) in *A*.
20 R/S	C?*value*	Stores 20 in *B*.
2 R/S	X=4.1751	Stores 2 in *C* and calculates X (which in this case is also known as *T*).
R/S	X=-0.0977	Calculates the other root.

Note that since a negative time has no meaning in the context of this problem, the first result, 4.1751 seconds, is the meaningful answer.

Example 4. Find the roots of the following second-degree polynomial using the program as it is listed. Then change the sense of the comparison at line Q12 so that the second root is computed first and then the results are compared. Remember to restore the original line or clear the program when you finish this example.

$$x^2 + (3 \times 10^6)x + 1 = 0$$

Keys:	Display:	Description:
XEQ Q	A?*value*	Starts program.
1 R/S	B?*value*	Stores 1 in *A*.
3 E 6 R/S	C?*value*	Stores 3×10^6 in *B*.
1 R/S	X=-3,000,000.00	Stores 1 in *C* and calculates the first root.

R/S	X=-3.3333E-7	Calculates the second root.
■ PRGM	Q36 GTO Q	Switches to program entry.
■ GTO . Q12	Q12 x<0?	Moves program pointer to line Q12.
←	Q11 CF 0	Deletes line Q12.
■ TESTS {x?0} {>0}	Q12 x>0?	Adds the conditional test $x>0$?.
C	-3.3333E-7	Cancels program entry.
XEQ Q	A?1.0000	Starts program.
R/S R/S		Skips data entry since values are already stored.
R/S	X=0.0000	Calculates first root using previous inputs.
R/S	DIVIDE BY 0	Attempts to compute second root.

As you can see, the results of a simple change in the order of calculation can be quite significant.

If you substitute the first values calculated back into the equation, you will find that the left-hand side of the equation is zero for the root of smaller absolute value (as it theoretically should be) and 1 for the root of larger absolute value. Does this mean that the result of −3,000,000.0000 is incorrect? The answer to this question is a qualified no. If you increment or decrement this value by one count in the least significant digit and substitute the result back into the original equation, then the left-hand side will be 31 or −29. Thus, −3,000,000.0000, while not being exactly correct, is the *best possible 12-digit result* that could be generated.

Coordinate Transformations

This program provides two-dimensional coordinate translation and rotation.

The following formulas are used to convert a point P from the Cartesian coordinate pair (x, y) in the old system to the pair (u, v) in the new, translated, rotated system.

$$u = (x - m)\cos\theta + (y - n)\sin\theta$$

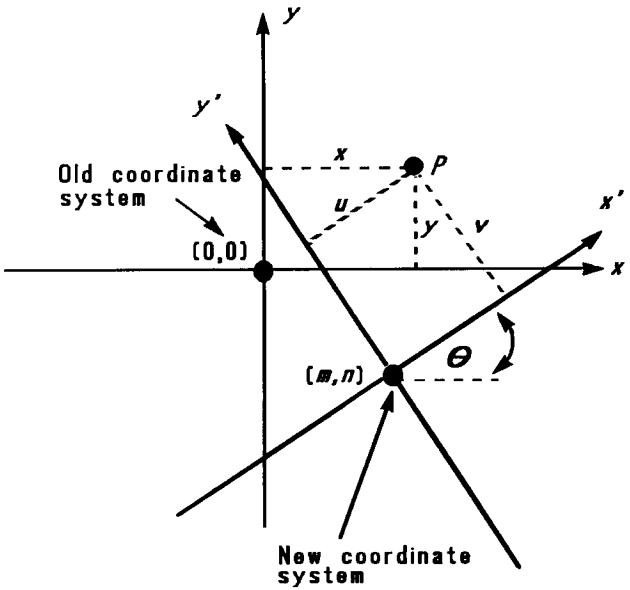
$$v = (y - n)\cos\theta - (y - n)\sin\theta$$

The inverse transformation is accomplished with the formulas below.

$$x = u\cos\theta - v\sin\theta + m$$

$$y = u\sin\theta + v\cos\theta + n$$

The HP-32S complex and polar-to-rectangular functions make these computations straightforward.



A Two-Dimensional Rotation About the Axis

Program Listing:

Program Lines:	Description:
D01 LBL D	This routine defines the new coordinate system.
D02 INPUT M	Prompts for and stores M, the new origin's x-coordinate.
D03 INPUT N	Prompts for and stores N, the new origin's y-coordinate.
D04 INPUT T	Prompts for and stores T, the angle θ.
D05 GTO D	Loops for review of inputs.

Bytes and Checksum: 007.5, 1CD9

N01 LBL N	This routine converts from the old system to the new system.
N02 INPUT X	Prompts for and stores X, the old x-coordinate.
N03 INPUT Y	Prompts for and stores Y, the old y-coordinate.
N04 RCL X	Pushes Y up and recalls X to the X-register.
N05 RCL N	Pushes X and Y up and recalls N to the X-register.
N06 RCL M	Pushes N, X, and Y up and recalls M.
N07 CMPLX-	Calculates $(X-M)$ and $(Y-N)$.
N08 RCL T	Pushes $(X-M)$ and $(Y-N)$ up and recalls T.
N09 +/-	Changes the sign of T because $\sin(-T)$ equals $-\sin(T)$.
N10 1	Sets radius to 1 for computation of $\cos(T)$ and $-\sin(T)$.
N11 θ,r→y,x	Calculates $\cos(T)$ and $-\sin(T)$ in X- and Y-registers.
N12 CMPLX×	Calculates $(X-M)\cos(T) + (Y-N)\sin(T)$ and $(Y-N)\cos(T) - (X-M)\sin(T)$.
N13 STO U	Stores x-coordinate in variable U.
N14 x<>y	Swaps positions of the coordinates.
N15 STO V	Stores y-coordinate in variable V.
N16 x<>y	Swaps positions of coordinates back.
N17 VIEW U	Halts program to display U.
N18 VIEW V	Halts program to display V.
N19 GTO N	Goes back for another calculation.

Bytes and Checksum: 028.5, 6078

001 LBL 0	This routine converts from the new system to the old system.
002 INPUT U	Prompts for and stores U.
003 INPUT V	Prompts for and stores V.
004 RCL U	Pushes V up and recalls U.
005 RCL T	Pushes U and V up and recalls T.
006 1	Sets radius to 1 for the computation of $\sin(T)$ and $\cos(T)$.
007 θ,r→y,x	Calculates $\cos(T)$ and $\sin(T)$.
008 CMPLX×	Calculates $U\cos(T) - V\sin(T)$ and $U\sin(T) + V\cos(T)$.
009 RCL N	Pushes up previous results and recalls N.
010 RCL M	Pushes up results and recalls M.
011 CMPLX+	Completes calculation by adding M and N to previous results.
012 STO X	Stores the x-coordinate in variable X.
013 x<>y	Swaps the positions of the coordinates.
014 STO Y	Stores the y-coordinate in variable Y.
015 x<>y	Swaps the positions of the coordinates back.
016 VIEW X	Halts the program to display X.
017 VIEW Y	Halts the program to display Y.
018 GTO 0	Goes back for another calculation.

Bytes and Checksum: 027.0, 9AE6

Flags Used: None.

Memory Required: 119 bytes: 63 for program, 56 for variables.

Program Instructions:

1. Key in the program routines; press [C] when done.
2. Press [XEQ] D to start the prompt sequence which defines the coordinate transformation.
3. Key in the x-coordinate of the origin of the new system M and press [R/S].
4. Key in the y-coordinate of the origin of the new system N and press [R/S].

5. Key in the rotation angle T and press [R/S].
6. To translate from the old system to the new system, continue with instruction step 7. To translate from the new system to the old system, skip to step 12.
7. Press [XEQ] N to start the old-to-new transformation routine.
8. Key in X and press [R/S].
9. Key in Y, press [R/S], and see the x-coordinate, U, in the new system.
10. Press [R/S] and see the y-coordinate, V, in the new system.
11. For another old-to-new transformation, press [R/S] and go to step 8. For a new-to-old transformation, continue with step 12.
12. Press [XEQ] O to start the new-to-old transformation routine.
13. Key in U (the x-coordinate in the new system) and press [R/S].
14. Key in V (the y-coordinate in the new system) and press [R/S] to see X.
15. Press [R/S] to see Y.
16. For another new-to-old transformation, press [R/S] and go to step 13. For an old-to-new transformation, go to step 7.

Variables Used:

M The x-coordinate of the origin of the new system.

N The y-coordinate of the origin of the new system.

T The rotation angle, θ, between the old and new systems.

X The x-coordinate of a point in the old system.

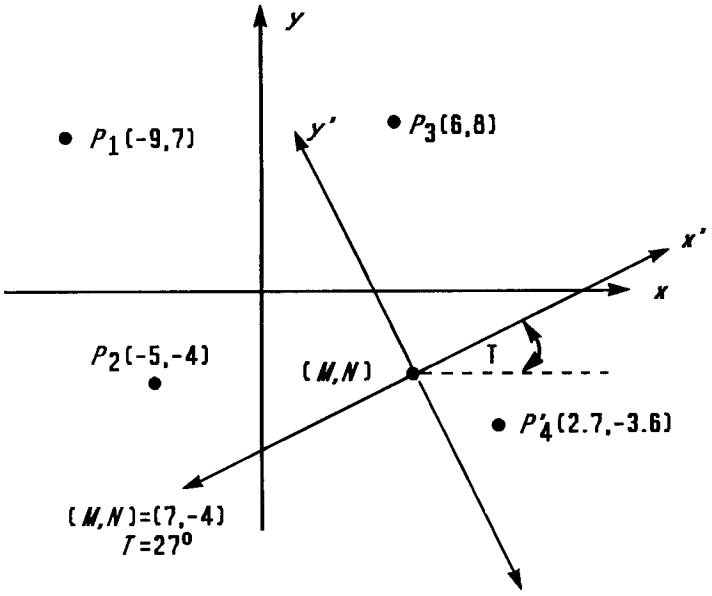
Y The y-coordinate of a point in the old system.

U The x-coordinate of a point in the new system.

V The y-coordinate of a point in the new system.

Remarks: For translation only, key zero for T. For rotation only, key zero for M and N.

Example: For the coordinate systems shown below, convert points P_1, P_2, and P_3, which are currently in the (X,Y) system, to points in the (X',Y') system. Convert point P'_4, which is in the (X',Y') system, to the (X,Y) system.



Keys:	Display:	Description:
■ MODES {DG}		Sets Degrees mode since T is given in degrees.
XEQ D	M?*value*	Starts the routine that defines the transformation.
7 R/S	N?*value*	Stores 7 in M.
4 +/- R/S	T?*value*	Stores -4 in N.
27 R/S	M?7.0000	Stores 27 in T.

XEQ N	X?*value*	Starts the old-to-new routine.
9 +/- R/S	Y?*value*	Stores −9 in *X*.
7 R/S	U=-9.2622	Stores 7 in *Y* and calculates *U*.
R/S	V=17.0649	Calculates *V*.
R/S	X?-9.0000	Resumes the old-to-new routine for next problem.
5 +/- R/S	Y?7.0000	Stores −5 in *X*.
4 +/- R/S	U=-10.6921	Stores −4 in *Y*.
R/S	V=5.4479	Calculates *V*.
R/S	X?-5.0000	Resumes the old-to-new routine for next problem.
6 R/S	Y?-4.0000	Stores 6 in *X*.
8 R/S	U=4.5569	Stores 8 in *Y* and calculates *U*.
R/S	V=11.1461	Calculates *V*.
XEQ O	U?4.5569	Starts the new-to-old routine.
2.7 R/S	V?11.1461	Stores 2.7 in *U*.
3.6 +/- R/S	X=11.0401	Stores −3.6 in *V* and calculates *X*.
R/S	Y=-5.9818	Calculates *Y*.

13

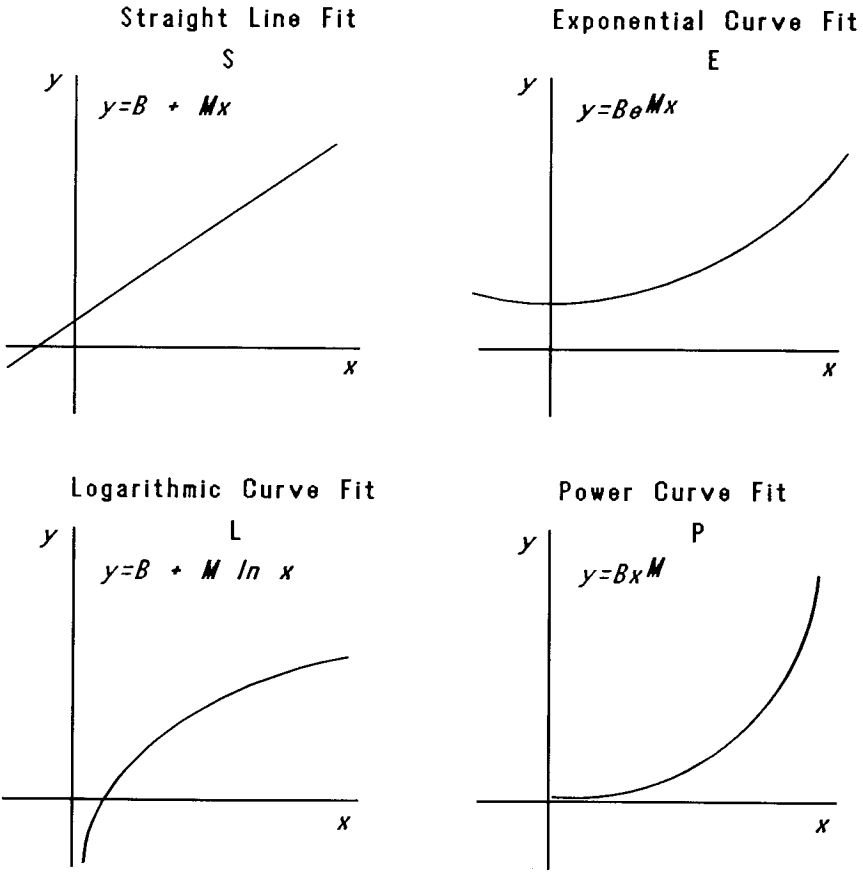
Statistics Programs

The memory usage and checksum for each program label can be checked using the catalog of programs (page 85).

Curve Fitting

This program can be used to fit one of four models of equations to your data. These models are the straight line, the logarithmic curve, the exponential curve and the power curve. The program accepts two or more (x, y) data pairs and then calculates the correlation coefficient, r, and the two regression coefficients, m and b. The program includes a routine to calculate the estimates $\hat{x}$ and $\hat{y}$. (For definitions of these values, see "Linear Regression" in chapter 11.)

Samples of the curves and the relevant equations are shown below. The internal regression functions of the HP-32S are used to compute the regression coefficients.



To fit logarithmic curves, values of x must be positive. To fit exponential curves, values of y must be positive. To fit power curves, both x and y must be positive. A LOG(NEG) error will occur if a negative number is entered for these cases.

Data values of large magnitude but relatively small differences can incur problems of precision, as can data values of greatly different magnitudes. Refer to "Limitations in Precision of Data" in chapter 11.

Program Listing:

Program Lines:	Description:
S01 LBL S	This routine sets the status for the straight-line model.
S02 1	Enters index value for later storage in *i* (for indirect addressing).
S03 CF 0	Clears flag 0, the indicator for lnX.
S04 CF 1	Clears flag 1, the indicator for lnY.
S05 GTO Z	Branches to common entry point Z.

Bytes and Checksum: 007.5, 17CA

L01 LBL L	This routine sets the status for the logarithmic model.
L02 2	Enters index value for later storage in *i* (for indirect addressing).
L03 SF 0	Sets flag 0, the indicator for lnX.
L04 CF 1	Clears flag 1, the indicator for lnY.
L05 GTO Z	Branches to common entry point Z.

Bytes and Checksum: 007.5, 6047

E01 LBL E	This routine sets the status for the exponential model.
E02 3	Enters index value for later storage in *i* (for indirect addressing).
E03 CF 0	Clears flag 0, the indicator for lnX.
E04 SF 1	Sets flag 1, the indicator for lnY.
E05 GTO Z	Branches to common entry point Z.

Bytes and Checksum: 007.5, C0F1

P01 LBL P	This routine sets the status for the power model.
P02 4	Enters index value for later storage in *i* (for indirect addressing).
P03 SF 0	Sets flag 0, the indicator for lnX.
P04 SF 1	Sets flag 1, the indicator for lnY.

Bytes and Checksum: 006.0, A26B

```
Z01 LBL Z          Defines common entry point for all models.
Z02 CLΣ            Clears the statistics registers.
Z03 STO i          Stores the index value in i for indirect
                   addressing.
Z04 0              Sets the loop counter to zero for the first
                   input.
```
Bytes and Checksum: 006.0, CC1B

```
W01 LBL W          Defines the beginning of the input loop.
W02 1              Adjusts the loop counter by one to prompt
                   for input.
W03 +
W04 STO X          Stores loop counter in X so that it will ap-
                   pear with the prompt for X.
W05 INPUT X        Displays counter with prompt and stores X
                   input.
W06 FS? 0          If flag 0 is set...
W07 LN             ...takes the natural log of the input.
W08 STO B          Stores that value for the correction routine.
W09 INPUT Y        Prompts for and stores Y.
W10 FS? 1          If flag 1 is set...
W11 LN             ...takes the natural log of the input.
W12 STO R
W13 RCL B
W14 Σ+             Accumulates B and R as x,y-data pair in sta-
                   tistics registers.
W15 GTO W          Loops for another X, Y pair.
```
Bytes and Checksum: 022.5, 1A43

```
U01 LBL U          Defines the beginning of the "undo"
                   routine.
U02 RCL R          Recalls the most recent data pair.
U03 RCL B
U04 Σ-             Deletes this pair from the statistical
                   accumulation.
U05 GTO W          Loops for another X,Y pair.
```
Bytes and Checksum: 007.5, 5D02

```
R01 LBL R          Defines the start of the output routine.
R02 r              Calculates the correlation coefficient.
R03 STO R          Stores it in R.
R04 VIEW R         Displays the correlation coefficient.
R05 b              Calculates the coefficient b.
R06 FS? 1          If flag 1 is set, takes the natural antilog of b.
R07 e^x
R08 STO B          Stores b in B.
R09 VIEW B         Displays value.
R10 m              Calculates coefficient m.
R11 STO M          Stores m in M.
R12 VIEW M         Displays value.
```

Bytes and Checksum: 018.0, 7492

```
Y01 LBL Y          Defines the beginning of the estimation
                   (projection) loop.
Y02 INPUT X        Displays, prompts for, and, if changed,
                   stores x-value in X.
Y03 XEQ(i)         Calls subroutine to compute ŷ.
Y04 STO Y          Stores ŷ-value in Y.
Y05 INPUT Y        Displays, prompts for, and, if changed,
                   stores y-value in Y.
Y06 6
Y07 STO+ i         Adjusts index value to address the appropri-
                   ate subroutine.
Y08 XEQ(i)         Calls subroutine to compute x̂.
Y09 STO X          Stores x̂ in X for next loop.
Y10 GTO Y          Loops for another estimate.
```

Bytes and Checksum: 015.0, 9AEA

```
A01 LBL A          This subroutine calculates ŷ for the straight-
                   line model.
A02 RCL M
A03 RCL× X
A04 RCL+ B         Calculates ŷ = MX + B.
A05 RTN            Returns to the calling routine.
```

Bytes and Checksum: 007.5, 0E85

```
G01 LBL G
```
This subroutine calculates $\hat{x}$ for the straight-line model.

```
G02 STO- i
```
Restores index value to its original value.

```
G03 RCL Y
G04 RCL- B
G05 RCL÷ M
```
Calculates $\hat{x} = (Y - B) \div M$.

```
G06 RTN
```
Returns to the calling routine.

Bytes and Checksum: 009.0, FDF1

```
B01 LBL B
```
This subroutine calculates $\hat{y}$ for the logarithmic model.

```
B02 RCL X
B03 LN
B04 RCL× M
B05 RCL+ B
```
Calculates $\hat{y} = M \ln X + B$.

```
B06 RTN
```
Returns to the calling routine.

Bytes and Checksum: 009.0, 1B06

```
H01 LBL H
```
This subroutine calculates $\hat{x}$ for the logarithmic model.

```
H02 STO- i
```
Restores index value to its original value.

```
H03 RCL Y
H04 RCL- B
H05 RCL÷ M
H06 e^x
```
Calculates $\hat{x} = e^{(Y - B) \div M}$.

```
H07 RTN
```
Returns to the calling routine.

Bytes and Checksum: 010.5, C783

```
C01 LBL C
```
This subroutine calculates $\hat{y}$ for the exponential model.

```
C02 RCL M
C03 RCL× X
C04 e^x
C05 RCL× B
```
Calculates $\hat{y} = Be^{MX}$.

```
C06 RTN
```
Returns to the calling routine.

Bytes and Checksum: 009.0, B411

```
I01 LBL I          This subroutine calculates x̂ for the ex-
                   ponential model.
I02 STO- i         Restores index value to its original value.
I03 RCL Y
I04 RCL÷ B
I05 LN
I06 RCL÷ M         Calculates x̂ = (ln(Y ÷ B)) ÷ M.
I07 RTN            Returns to the calling routine.
```
Bytes and Checksum: 010.5, 01D6

```
D01 LBL D          This subroutine calculates ŷ for the power
                   model.
D02 RCL X
D03 RCL M
D04 y^x
D05 RCL× B         Calculates Y = B(X^M).
D06 RTN            Returns to the calling routine.
```
Bytes and Checksum: 009.0, B4D4

```
J01 LBL J          This subroutine calculates x̂ for the power
                   model.
J02 STO- i         Restores index value to its original value.
J03 RCL Y
J04 RCL÷ B
J05 RCL M
J06 1/x
J07 y^x            Calculates x̂ = (Y/B)^(1/M).
J08 RTN            Returns to the calling routine.
```
Bytes and Checksum: 012.0, FAA4

Flags Used: Flag 0 is set if a natural log is required of the *X* input. Flag 1 is set if a natural log is required of the *Y* input.

Memory Required: 270 bytes: 174 for program, 96 for data (statistics registers 48).

Program Instructions:

1. Key in the program routines; press [C] when done.
2. Press [XEQ] and select the type of curve you wish to fit by pressing:
   - S for a straight line;
   - L for a logarithmic curve;
   - E for an exponential curve; or
   - P for a power curve.
3. Key in x-value and press [R/S].
4. Key in y-value and press [R/S].
5. Repeat steps 3 and 4 for each data pair. If you discover that you have made an error after you have pressed [R/S] in step 3 (with the Y?*value* prompt still visible), press [R/S] again (displaying the X?*value* prompt) and press [XEQ] U to *undo* (remove) the last data pair. If you discover that you made an error after step 4, press [XEQ] U. In either case continue at step 3.
6. After all data are keyed in, press [XEQ] R to see the correlation coefficient, R.
7. Press [R/S] to see the regression coefficient B.
8. Press [R/S] to see the regression coefficient M.
9. Press [R/S] to see the X?*value* prompt for the $\hat{x}$-, $\hat{y}$-estimation routine.
10. If you wish to estimate $\hat{y}$ based on x, key in x at the X?*value* prompt, then press [R/S] to see $\hat{y}$ (Y?).
11. If you wish to estimate $\hat{x}$ based on y, press [R/S] until you see the Y?*value* prompt, key in y, then press [R/S] to see $\hat{x}$ (X?).
12. For more estimations, go to steps 10 or 11.
13. For a new case, go to step 2.

Variables Used:

B	Regression coefficient (y-intercept of a straight line); also used for scratch.
M	Regression coefficient (slope of a straight line).
R	Correlation coefficient; also used for scratch.
X	The x-value of a data pair when entering data; the hypothetical x when projecting $\hat{y}$; or $\hat{x}$ (x-estimate) when given a hypothetical y.
Y	The y-value of a data pair when entering data; the hypothetical y when projecting $\hat{x}$; or $\hat{y}$ (y-estimate) when given a hypothetical x.
i	Index variable used to indirectly address the correct $\hat{x}$-, $\hat{y}$-projection equation.
Statistics registers	Statistical accumulation and computation.

Example 1. Fit a straight line to the data below. Make an intentional error when keying in the third data pair and correct it with the undo routine. Also, estimate y for an x value of 37. Estimate x for a y value of 101.

X	40.5	38.6	37.9	36.2	35.1	34.6
Y	104.5	102	100	97.5	95.5	94

Keys:	**Display:**	**Description:**
[XEQ] S	X?1.0000	Starts straight-line routine.
40.5 [R/S]	Y?*value*	Enters x-value of data pair.
104.5 [R/S]	X?2.0000	Enters y-value of data pair.

38.6 [R/S]	Y?104.5000	Enters x-value of data pair.
102 [R/S]	X?3.0000	Enters y-value of data pair.

Now intentionally enter 379 instead of 37.9 so that you can see how to correct incorrect entries.

379 [R/S]	Y?102.0000	Enters wrong x-value of data pair.
[R/S]	X?4.0000	Retrieves X? prompt.
[XEQ] U	X?3.0000	Deletes the last pair.

Now proceed with the correct data entry.

37.9 [R/S]	Y?102.0000	Enters correct x-value of data pair.
100 [R/S]	X?4.0000	Enters y-value of data pair.
36.2 [R/S]	Y?100.0000	Enters x-value of data pair.
97.5 [R/S]	X?5.0000	Enters y-value of data pair.
35.1 [R/S]	Y?97.5000	Enters x-value of data pair.
95.5 [R/S]	X?6.0000	Enters y-value of data pair.
34.6 [R/S]	Y?95.5000	Enters x-value of data pair.
94 [R/S]	X?7.0000	Enters y-value of data pair.
[XEQ] R	R=0.9955	Calculates the correlation coefficient.

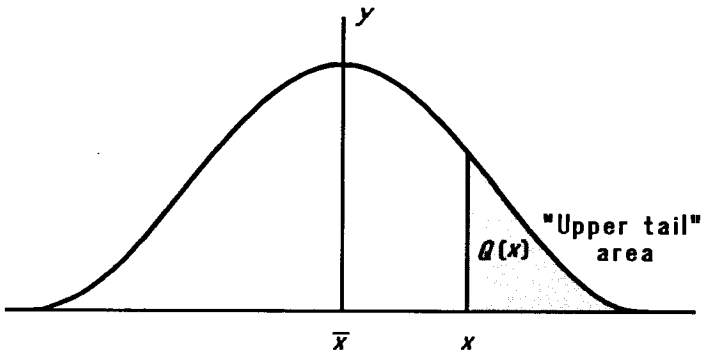
R/S	B=33.5271	Calculates regression coefficient B.
R/S	M=1.7601	Calculates regression coefficient M.
R/S	X?7.0000	Prompts for hypothetical x-value.
37 R/S	Y?98.6526	Stores 37 in X and calculates $\hat{y}$.
101 R/S	X?38.3336	Stores 101 in Y and calculates $\hat{x}$.

Example 2. Repeat example 1 (using the same data) for logarithmic, exponential and power curve fits. The table below gives you the starting execution label and the results (the correlation and regression coefficients and the x- and y-estimates) for each type of curve. You will need to reenter the data values each time you run the program for a different curve fit.

	Logarithmic	**Exponential**	**Power**
To start:	XEQ L	XEQ E	XEQ P
R	0.9965	0.9945	0.9959
B	−139.0088	51.1312	8.9730
M	65.8446	0.0177	0.6640
Y ($\hat{y}$ when $X=37$)	98.7508	98.5870	98.6845
X ($\hat{x}$ when $Y=101$)	38.2857	38.3628	38.3151

Normal and Inverse-Normal Distributions

Normal distribution is frequently used to model the behavior of random variation about a mean. This model assumes that the sample distribution is symmetric about the mean, M, with a standard deviation, S, and approximates the shape of the bell-shaped curve shown below. Given a value x, this program calculates the probability that a random selection from the sample data will have a higher value. This is known as the upper tail area, $Q(x)$. This program also provides the inverse: given a value $Q(x)$, the program calculates the corresponding value x.



$$Q(x) = 0.5 - \frac{1}{\sigma\sqrt{2\pi}} \int_{\bar{x}}^{x} e^{-((x - \bar{x}) \div \sigma)^2 \div 2}\, dx.$$

This program uses the built-in integration feature of the HP-32S to integrate the equation of the normal frequency curve. The inverse is obtained using Newton's method to iteratively search for a value of x which yields the given probability $Q(x)$.

Program Listing:

Program Lines:	**Description:**
S01 LBL S	This routines initializes the standard-deviation program.
S02 0	Stores default value for mean.
S03 STO M	
S04 INPUT M	Prompts for and stores mean, *M*.

```
S05 1                    Stores default value for standard deviation.
S06 STO S
S07 INPUT S              Prompts for and stores standard deviation,
                         S.
S08 RTN                  Stops displaying value of standard
                         deviation.
```

Bytes and Checksum: 012.0, 1F60

```
D01 LBL D                This routine calculates Q(X) given X.
D02 INPUT X              Prompts for and stores X.
D03 XEQ Q                Calculates upper tail area.
D04 STO Q                Stores value in Q so VIEW function can dis-
                         play it.
D05 VIEW Q               Displays Q(X).
D06 GTO D                Loops to calculate another Q(X).
```

Bytes and Checksum: 009.0, 002C

```
I01 LBL I                This routine calculates X given Q(X).
I02 INPUT Q              Prompts for and stores Q(X).
I03 RCL M                Recalls the mean.
I04 STO X                Stores the mean as the guess for X, called
                         X_guess.
```

Bytes and Checksum: 006.0, ED6E

```
T01 LBL T                This label defines the start of the iterative
                         loop.
T02 XEQ Q                Calculates (Q(X_guess) − Q(X)).
T03 RCL- Q
T04 RCL X
T05 STO D
T06 R↓
T07 XEQ F                Calculates the derivative at X_guess.
T08 RCL÷ T
T09 ÷                    Calculates the correction for X_guess.
T10 STO+ X               Adds the correction to yield a new X_guess.
T11 ABS
T12 0.0001
T13 x<y?                 Tests to see if the correction is significant.
T14 GTO T                Goes back to start of loop if correction is
                         significant. Continues if correction is not
                         significant.
T15 RCL X
```

```
T16 VIEW X          Displays the calculated value of X.
T17 GTO I           Loops to calculate another X.
```
Bytes and Checksum: 033.5, 4355

```
Q01 LBL Q           This subroutine calculates the upper-tail
                    area Q(x).
Q02 RCL M           Recalls the lower limit of integration.
Q03 RCL X           Recalls the upper limit of integration.
Q04 FN= F           Selects the function defined by LBL F for
                    integration.
Q05 ∫FN d D         Integrates the normal function using the
                    dummy variable D.
Q06 2
Q07 π
Q08 ×
Q09 SQRT
Q10 RCL S
Q11 ×               Calculates S × √(2π).
Q12 STO T           Stores result temporarily for inverse routine.
Q13 ÷
Q14 +/-
Q15 0.5
Q16 +               Adds half the area under the curve since we
                    integrated using the mean as the lower limit.
Q17 RTN             Returns to the calling routine.
```
Bytes and Checksum: 033.5, 4B20

```
F01 LBL F           This subroutine calculates the integrand for
                    the normal function e^-((X-M)÷S)^2÷2.
F02 RCL D
F03 RCL- M
F04 RCL÷ S
F05 x²
F06 2
F07 ÷
F08 +/-
F09 e^x
F10 RTN             Returns to the calling routine.
```
Bytes and Checksum: 015.0, 034D

Flags Used: None.

Memory Required: 157 bytes: 109 for program, 48 for variables.

Remarks: The accuracy of this program is dependent on the display setting. For inputs in the range between ±3 standard deviations, a display of four or more significant figures is adequate for most applications. At full precision, the input limit becomes ±5 standard deviations. Computation time is significantly less with a lower number of displayed digits.

In routine N, the constant 0.5 may be replaced by 2 and [1/x]. This will save 6.5 bytes at the expense of clarity.

You do not need to key in the inverse routine (in routines I and T) if you are not interested in the inverse capability.

Program Instructions:

1. Key in the program routines; press [C] when done.
2. Press [XEQ] S.
3. After the prompt for *M*, key in the population mean and press [R/S]. (If the mean is zero, just press [R/S].)
4. After the prompt for *S*, key in the population standard deviation and press [R/S]. (If the standard deviation is 1, just press [R/S].)
5. To calculate *X* given *Q*(X), skip to step 9 of these instructions.
6. To calculate *Q*(X) given *X*, [XEQ] D.
7. After the prompt, key in the value of *X* and press [R/S]. The result, *Q*(X), is displayed.
8. To calculate *Q*(X) for a new *X* with the same mean and standard deviation, press [R/S] and go to step 7.
9. To calculate *X* given *Q*(X), press [XEQ] I.
10. After the prompt, key in the value of *Q*(X) and press [R/S]. The result, *X*, is displayed.
11. To calculate *X* for a new *Q*(X) with the same mean and standard deviation, press [R/S] and go to step 10.

Variables Used:

D Dummy variable of integration.

M Population mean, default value zero.

Q Probability corresponding to the upper-tail area.

S Population standard deviation, default value of 1.

T Variable used temporarily to pass the value $S\sqrt{2\pi}$ to the inverse program.

X Input value that defines the left side of the upper-tail area.

Example 1. Your good friend informs you that your blind date has "3σ" intelligence. You interpret this to mean that this person is more intelligent than the local population except for people more than three standard deviations above the mean. Suppose that you intuit that the local population contains 10,000 possible blind dates. How many people fall into the "3σ" band? Since this problem is stated in terms of standard deviations, use the default values of zero for *M* and 1 for *S*.

Keys:	**Display:**	**Description:**
[XEQ] S	M?0.0000	Starts the initialization routine.
[R/S]	S?1.0000	Accepts the default value of zero for *M*.
[R/S]	1.0000	Accepts the default value of 1 for *S*.
[XEQ] D	X?*value*	Starts the distribution program and prompts for *X*.
3 [R/S]	Q=0.0014	Enters 3 for *X* and starts computation of *Q*(X). Displays the ratio of the population smarter than everyone within three standard deviations of the mean.

10000 [x]	13.5049	Multiplies by the population. Displays the approximate number of blind dates in the local population, which meet the criteria.

Since your friend has been known to exaggerate from time to time, you decide to see how rare a "2σ" date might be. Note that the program may be rerun simply by pressing [R/S].

Keys:	**Display:**	**Description:**
[R/S]	X?3.0000	Resumes program.
2 [R/S]	Q=0.0227	Enters X-value of 2 and calculates $Q(X)$.
10000 [x]	227.4937	Multiplies by the population for the revised estimate.

Example 2.. The mean of a set of test scores is 55. The standard deviation is 15.3. Assuming that the standard normal curve adequately models the distribution, what is the probability that a randomly selected student scored 90 or above? What is the score that only 10 percent of the students would be expected to have surpassed? What would be the score that only 20 percent of the students would have failed to achieve?

Keys:	Display:	Description:
[XEQ] S	M?0.0000	Starts the initialization routine.
55 [R/S]	S?1.0000	Stores 55 for the mean.
15.3 [R/S]	15.3000	Stores 15.3 for the standard deviation.
[XEQ] D	X?*value*	Starts the distribution program and prompts for X.
90 [R/S]	Q=0.0111	Enters 90 for X and calculates Q(X).

Thus, we would expect that only about 1 percent of the students would do better than score 90.

Keys:	Display:	Description:
[XEQ] I	Q?0.0111	Starts the inverse routine.
.1 [R/S]	X=74.6078	Stores 0.1 (10 percent) in $Q(X)$ and calculates X.
[R/S]	Q?0.1000	Resumes the inverse routine.
.8 [R/S]	X=42.1232	Stores 0.8 (100 percent minus 20 percent) in $Q(X)$ and calculates X.

14

Miscellaneous Programs

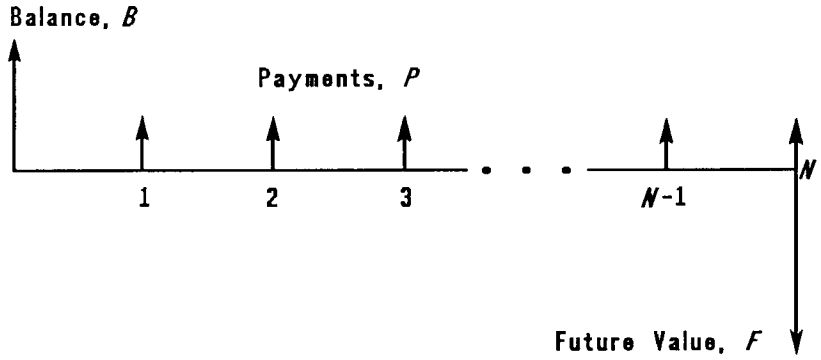
The memory usage and checksum for each program label can be checked using the catalog of programs (page 85).

Time Value of Money

Given four of the five values of the time-value-of-money equation, this program solves for the fifth. It is useful in a wide variety of financial applications such as consumer and home loans and savings accounts.

The equation used to solve problems for the time value of money is:

$$P\left[\frac{1-(1+Z)^{-N}}{Z}\right]+F(1+Z)^{-N}+B=0.$$



A Cash-Flow Diagram

The signs of the cash values (balance, *B;* payment, *P;* and future balance, *F*) correspond to the direction that the cash flows. Money that you receive has a positive sign while money that you pay has a negative sign. Note that any problem can be viewed from two perspectives. The lender and the borrower view the same problem with reversed signs.

Program Listing:

Program Lines:	**Description:**
N01 LBL N	This routine calculates the number of payments, *N*.
N02 14	Enters the number that corresponds to N for indirect addressing.
N03 GTO L	Branches to the common control routine, L.
Bytes and Checksum: 004.5, 61E5	
I01 LBL I	This routine calculates the interest rate, *I*.
I02 9	Enters the number that corresponds to I for indirect addressing.
I03 GTO L	Branches to the common control routine, L.
Bytes and Checksum: 004.5, DA04	
B01 LBL B	This routine calculates the initial balance, *B*.
B02 2	Enters the number that corresponds to B for indirect addressing.
B03 GTO L	Branches to the common control routine, L.
Bytes and Checksum: 004.5, 98EB	
P01 LBL P	This routine calculates the periodic payment, *P*.
P02 16	Enters the number that corresponds to P for indirect addressing.
P03 GTO L	Branches to the common control routine, L.
Bytes and Checksum: 004.5, A556	
F01 LBL F	This routine calculates the future value, *F*.
F02 6	Enters the number that corresponds to F for indirect addressing.
Bytes and Checksum: 003.0, 6779	

L01 LBL L	This label controls the computation of the selected variable.
L02 STO i	Stores the index value (for indirect addressing) in *i*.
L03 FN= T	Selects the T routine, which contains the equation, for SOLVE.
L04 SOLVE(i)	Solves for the variable indirectly addressed by *i*.
L05 VIEW(i)	Displays the result addressed by *i*.
L06 GTO(i)	Goes back for another calculation.

Bytes and Checksum: 009.0, 7878

T01 LBL T	This routine contains the equation defining the time value of money.
T02 INPUT N	Prompts for and stores *N*.
T03 INPUT I	Prompts for and stores *I*.
T04 INPUT B	Prompts for and stores *B*.
T05 INPUT P	Prompts for and stores *P*.
T06 INPUT F	Prompts for and stores *F*.
T07 RCL I	Recalls the interest rate in percent.
T08 x=0?	If $I=0\ldots$
T09 GTO K	...then uses the equation in routine K.
T10 100	
T11 ÷	
T12 STO Z	Converts *I* to decimal form and stores it in Z.
T13 1	
T14 +	Calculates $(1 + Z)$.
T15 RCL N	
T16 +/-	
T17 y^x	Calculates $(1 + Z)^{-N}$.
T18 ENTER	Duplicates quantity so that it can be used later.
T19 +/-	
T20 1	
T21 +	Calculates $1 - (1 + Z)^{-N}$.
T22 RCL÷ Z	Calculates $(1 - (1 + Z)^{-N}) \div Z$.

T23	RCL× P	Calculates $P\times(1-(1+Z)^{-N})\div Z$.
T24	x<>y	Swaps duplicate copy of $(1+Z)^{-N}$ (from line T18) into the X-register.
T25	RCL× F	Calculates $F\times(1+Z)^{-N}$.
T26	+	Calculates $F\times(1+Z)^{-N}+P\times(1-(1+Z)^{-N})\div Z$.
T27	RCL+ B	Calculates $F\times(1+Z)^{-N}+P\times(1-(1+Z)^{-N})\div Z+B$.
T28	RTN	Returns to calling routine.

Bytes and Checksum: 050.0, 429C

K01	LBL K	This routine is called if $I = 0$.
K02	RCL P	
K03	RCL× N	
K04	RCL+ F	
K05	RCL+ B	Calculates $P \times N + F + B$.
K06	RTN	Returns to calling routine.

Bytes and Checksum: 009.0, F2E0

Flags Used: None.

Memory Required: 145 bytes: 89 for program, 56 for variables.

Remarks: Since all of the computation for the program is done in routines T and K, it is possible to shorten the program by eliminating the other user-interface routines. To run the program in this shortened form, select the function defined by LBL T (■ SOLVE/∫ {FN=} T) and then solve for the variable you need (■ SOLVE/∫ {SOLVE} *variable*).

Program Instructions:

1. Key in the program routines; press C when done.
2. Select the appropriate routine:
   - XEQ N to calculate the number of compounding periods;
   - XEQ I to calculate the periodic interest;
   - XEQ B to calculate the initial balance of a loan or savings account;

- [XEQ] P to calculate the periodic payment;
- [XEQ] F to calculate the future value or balance of a loan.

3. Key in the values of the other four variables as they are prompted for, and press [R/S] after each value.
4. After the last [R/S], the result is displayed.
5. To recalculate the same variable using different data, press [R/S] and go to step 3.
6. For a totally new case go to step 2.

Variables Used:

N The number of compounding periods.

I The *periodic* interest rate as a percentage rate. (For example, if the *annual* interest rate is 15% but there are 12 payments per year, then the *periodic* interest rate is 15÷12=1.25%.)

Z The *periodic* interest rate as a decimal.

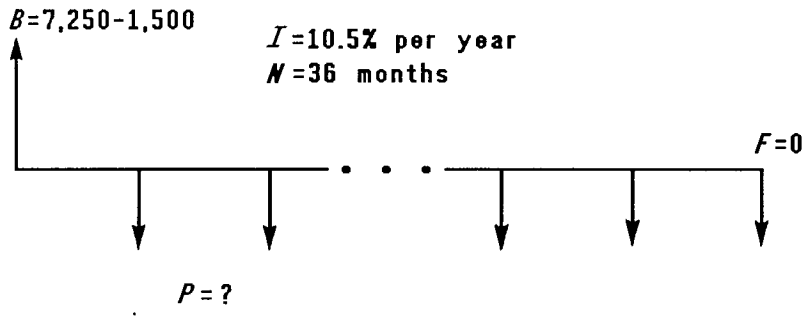
B The initial balance of loan or savings account.

P The periodic payment.

F The future value of a savings account or balance of a loan.

i The index variable, used here for indirect addressing.

Example: Part 1. You are financing the purchase of a car with a 3-year loan (36 months) at 10.5% annual interest compounded monthly. The purchase price of the car is $7,250. Your down payment is $1,500. What are your monthly payments?



Keys:	Display:	Description:
■ DISP {FX}2		Sets the display format to FIX 2.
XEQ P	N?*value*	Selects routine P, which calculates the periodic payment.
36 R/S	I?*value*	Stores 36 in *N*.
10.5 ENTER 12 ÷	I?0.88	Converts the annual interest rate to a monthly rate.
R/S	B?*value*	Stores the monthly interest rate in *I*.
7250 ENTER 1500 −	B?5,750.00	Calculates the beginning loan balance.
R/S	F?*value*	Sets *B* equal to the beginning balance.
0 R/S	P=-186.89	Stores zero in *F*, the future or ending balance, and calculates the payment of the loan.

The answer is negative since the loan has been viewed from the borrower's perspective. Money received by the borrower (the beginning balance) is positive, while money paid out is negative.

Part 2. What interest rate would reduce the monthly payment by $10?

Keys:	Display:	Description:
[XEQ] I	N?36.00	Selects routine I, which calculates the periodic interest rate.
[R/S]	B?5,750.00	Accepts 36 as the number of payments.
[R/S]	P?-186.89	Accepts $5,750.00 as the initial balance.
[ENTER]	P?-186.89	Copies payment in stack so that you can calculate with it. (The X-register will be overwritten by the next number entered.)
10 [+]	P?-176.89	Reduces the monthly payment by $10.00.
[R/S]	F?0.00	Stores the modified payment value.
[R/S]	I=0.56	Accepts zero as the future balance and calculates *I*, the monthly interest rate.
12 [×]	6.75	Calculates the annual interest rate.

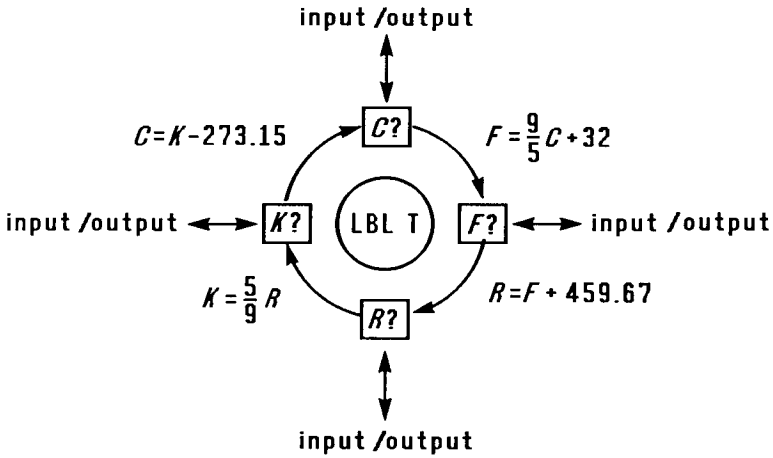
Part 3. Using the interest rate of 6.75%, assume that you sell the car after 2 years. What balance will you still owe? In other words, what is the future balance in 2 years?

Keys:	Display:	Description:
[XEQ] F	N?36.00	Selects routine F, which calculates the future value.
24 [R/S]	I?0.56	Changes the number of payments to 24 months.
[R/S]	B?5,750.00	Accepts the monthly interest rate.
[R/S]	P?-176.89	Accepts $5,750.00 as the initial balance.
[R/S]	F=-2,047.04	Accepts the payment value and calculates the future balance. Again, the sign is negative, indicating that you must pay out this money.

Unit Conversions

This program consists of two routines that convert one type of unit to another. One routine converts among Celsius, Fahrenheit, Rankine and Kelvin temperatures. The other routine converts among inches, feet, and meters, and among square inches, square feet and square meters. The programs can easily be modified to convert other types of units.

Both routines are based on the "Ferris wheel principle." The program has a circular structure. When the program is started, it loops through a cycle (or series) of input prompts. You repeatedly ignore the prompts (by pressing [R/S]) until the prompt that corresponds to the units of your input comes up. For instance, if you wanted to input kelvins, you would start the temperature program by pressing [XEQ] T and then by pressing [R/S] until the K? prompt appeared. At the prompt you would key in the temperature in kelvins and press [R/S] until you come to the prompt indicating the units that you desired (F? for Fahrenheit, for example). The value displayed with the prompt would be the temperature in degrees Fahrenheit. To end the program, press [C].



Ferris Wheel Structure for Temperature Conversion

The program has been designed to minimize the use of the stack. When the program is terminated, the values you had in the X- and Y-registers are left in the Y- and Z-registers, respectively, and the converted value is displayed. If the value that you wish to convert is in the X-register (the display) when you start the program, press [R↓] to retrieve it when you get to the correct prompt.

The length- and area-conversion routines give good examples of flag usage. Note that flag 2 was selected so that the **2** annunciator in the display would indicate that the unit is *squared.*

Program Listing:

Program Lines:	Description:
T01 LBL T	Starts the temperature-conversion routine.
T02 INPUT C	Displays the temperature in °C, or requests and stores Celsius input.
T03 9	
T04 ×	
T05 5	
T06 ÷	
T07 32	
T08 +	Converts from Celsius to Fahrenheit.
T09 STO F	Stores Fahrenheit temperature in *F*.
T10 R↓	Drops stack so that only one level is used.
T11 INPUT F	Displays temperature in °F, or requests and stores Fahrenheit input.
T12 459.67	
T13 +	Converts from Fahrenheit to Rankine.
T14 STO R	Stores Rankine temperature in *R*.
T15 R↓	Drops stack so that only one level is used.
T16 INPUT R	Displays temperature in °R, or requests and stores Rankine input.
T17 9	
T18 ÷	
T19 5	
T20 ×	Converts Rankine to Kelvin.
T21 STO K	Stores Kelvin temperature in *K*.
T22 R↓	Drops stack so that only one level is used.
T23 INPUT K	Displays temperature in kelvins, or requests and stores Kelvin input.
T24 273.15	
T25 −	Converts Kelvin to Celsius.
T26 STO C	Stores Celsius temperature in *C*.
T27 R↓	Drops stack so that only one level is used.
T28 GTO T	Loops back for more conversions.

Bytes and Checksum: 058.0, 9EF4

```
A01 LBL A          Starts the area-conversion routine.
A02 SF 2           Sets flag 2 to indicate area conversion
                   (squared length).
A03 GTO Q          Branches to the conversion routine.
```
Bytes and Checksum: 004.5, 2A52

```
L01 LBL L          Starts the length-conversion routine.
L02 CF 2           Clears the area-conversion flag.
```
Bytes and Checksum: 003.0, 2C3C

```
Q01 LBL Q          Starts the combined length- and area-con-
                   version routine.
Q02 INPUT I        Displays inches or square inches, or accepts
                   input.
Q03 12             Enters conversion factor for inches to feet.
Q04 FS? 2          Tests whether this is an area conversion.
Q05 x²             If yes, squares the conversion factor.
Q06 ÷              Calculates result.
Q07 STO F          Stores feet or square feet.
Q08 R↓             Drops stack so that only one level is used.
Q09 INPUT F        Displays feet or square feet, or accepts
                   input.
Q10 0.3048         Enters conversion factor for feet to meters.
Q11 FS? 2          Tests whether this is an area conversion.
Q12 x²             If yes, squares the conversion factor.
Q13 ×              Calculates the result.
Q14 STO M          Stores meters or square meters.
Q15 R↓             Drops stack so that only one level is used.
Q16 INPUT M        Displays meters or square meters, or accepts
                   input.
Q17 0.0254         Enters conversion factor for meters to
                   inches.
Q18 FS? 2          Tests whether this is an area conversion.
Q19 x²             If yes, squares the conversion factor.
Q20 ÷              Calculates the result.
Q21 STO I          Stores inches or square inches.
Q22 R↓             Drops stack so that only one level is used.
Q23 GTO Q          Loops back for more conversions.
```
Bytes and Checksum: 050.5, 2A15

Flags Used: Flag 2.

Memory Required: 164 bytes: 116 for program, 48 for variables.

Program Instructions:

1. Key in the program routines; press [C] when done.
2. Press [XEQ] and the appropriate label.
   - *A* for area conversion, or
   - *L* for length conversion, or
   - *T* for temperature conversion.
3. Press [R/S] until the appropriate input prompt appears.
4. Key in input (or press [R↓] to recover the input if it had been in the display when the routine was started).
5. Press [R/S] until the prompt (with a result) corresponds to the units you want to find.
6. Go to step 3 for another conversion.
7. Press [C] to clear the prompt and end the program.

Variables Used:

C Temperature in degrees Celsius.

F Temperature in degrees Fahrenheit; or feet.

R Temperature in degrees Rankine.

K Temperature in kelvins.

I Inches.

M Meters.

Example 1. Convert 212°F to kelvins.

Keys:	Display:	Description:
[XEQ] T	C?*value*	Selects the temperature conversion routine.
[R/S]	F?*value*	Searches for the Fahrenheit prompt.

212 [R/S]	R?671.6700	Enters Fahrenheit temperature and converts to degrees Rankine.
[R/S]	K?373.1500	Converts to kelvins.

Example 2. A floor measures 108 × 127 inches. How many square feet is this?

Keys:	**Display:**	**Description:**
108 [ENTER]		
127 [×]	K?13,716.0000	Calculates area in square inches.
[XEQ] A	I?*value*	Selects area-conversion routine.
[R↓]	I?13,716.0000	Rolls down from the Y-register the value you previously calculated for in^2.
[R/S]	F?95.2500	Calculates the square feet.
[C]	95.2500	Cancels the prompt and ends the program.

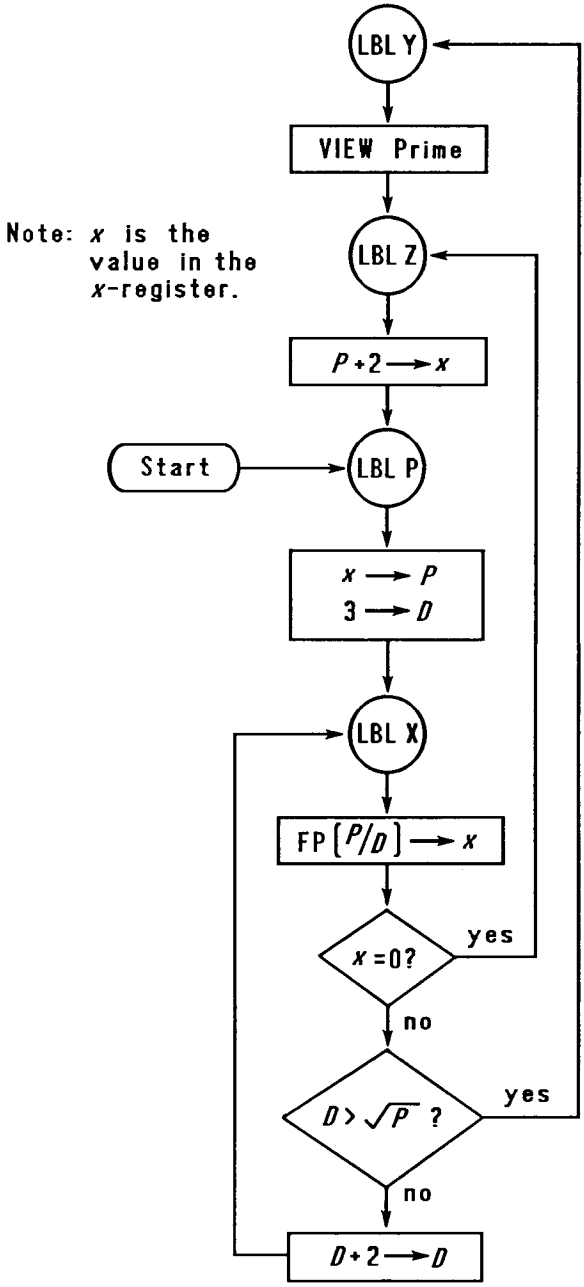
Example 3. Suppose that the result of a lengthy calculation is 3.787 and that it is currently in the display of your calculator. Further suppose that this value must be divided by a length specified in meters to complete the problem. You know that the length you need to divide by is 25.73 feet. Compute the final answer.

Keys:	Display:	Description:
3.787	3.787	Enters the hypothetical result.
[XEQ] L	I?*value*	Selects the length-conversion routine.
[R/S]	F?*value*	Moves to the prompt for feet.
25.73 [R/S]	M?7.8425	Enters the divisor in feet, then converts it to meters.
[C]	7.8425	Cancels the prompt and ends the program.
[÷]	0.4829	Calculates the final result.

Prime Number Generator

This program accepts any odd, positive integer greater than 3. If the number is a prime number (not evenly divisible by integers other than itself and 1), then the program returns the input value. If the input is not a prime number, then the program returns the first prime number larger than the input.

The program identifies non-prime numbers by exhaustively trying all possible factors. If a number is not prime, the program adds 2 (assuring that the value is still odd) and tests to see if it has found a prime. This process continues until a prime number is found.



Prime Number Flow Chart

Program Listing:

Program Lines:	Description:
Y01 LBL Y	This routine displays the prime number.
Y02 VIEW P	Displays the prime number *P*.
Bytes and Checksum: 003.0, 9D08	
Z01 LBL Z	This routine adds 2 to *P* before testing to see if *P* is prime.
Z02 2	
Z03 RCL+ P	
Bytes and Checksum: 004.5, E455	
P01 LBL P	This routine stores the input value for *P*.
P02 STO P	
P03 3	Stores 3 in test divisor, *D*.
P04 STO D	
Bytes and Checksum: 006.0, 9E38	
X01 LBL X	This routine tests *P* to see if it is prime.
X02 RCL P	
X03 RCL÷ D	
X04 FP	Finds the fractional part of $P \div D$.
X05 x=0?	Tests for a remainder of zero (number not prime).
X06 GTO Z	If the number is not prime, tries next possibility.
X07 RCL P	
X08 SQRT	
X09 RCL D	
X10 x>y?	Tests to see whether all possible factors have been tried.
X11 GTO Y	If all factors have been tried, branches to the display routine.
X12 2	Calculates the next possible factor, $D + 2$.
X13 STO+ D	

X14 GTO X	Branches to test the potential prime with the new factor.

Bytes and Checksum: 021.0, 43F2

Flags Used: None.

Memory Required: 50.5 bytes: 34.5 for program, 16 for variables.

Program Instructions:

1. Key in the program routines; press [C] when done.
2. Key in an odd integer.
3. Press [XEQ] *P* to start program. The prime number, *P*, will be displayed.
4. To see the next prime number, press [R/S].

Variables Used:

P Prime value and potential prime values.

D Divisor which is being used to test the current value of *P*.

Remarks: No tests are made to assure that the input is an odd, positive integer greater than 3.

Example. What is the first prime number after 789? What is the next prime number?

Keys:	Display:	Description:
789 [XEQ] P	P=797.0000	Keys in 789 and starts program; displays first prime number.
[R/S]	P=809.0000	Calculates the next prime number after 797.

Part 5

Appendixes and Reference

A

Assistance, Batteries, and Service

Obtaining Help in Operating the Calculator

We at Hewlett-Packard are committed to providing the owners of HP calculators with ongoing support. You can obtain answers to your questions about using the calculator from our Calculator Support Department.

We suggest that you read the next section, "Answers to Common Questions," before contacting us. Past experience has shown that many of our customers have similar questions about our products.

If you don't find an answer to your question, you can contact us using the address or phone number listed on the inside back cover.

Answers to Common Questions

Q. How can I determine if the calculator is operating properly?

A. Refer to page 246, which describes the diagnostic self-test.

Q. How do I change the number of decimal places in the display?

A. Use the ■ DISP function (page 30).

Q. My numbers contain commas instead of periods as decimal points. How do I restore the periods?

A. Use the ■ MODES function (page 29).

Q. How do I clear all or portions of memory?

A. ■[CLEAR] displays the CLEAR menu, which allows you to clear all variables, all programs (in program entry only), the statistics registers, or all of user memory (not during program entry).

Q. What does an "E" in a number (for example, 2.51E−13) mean?

A. *Exponent* of ten; that is, 2.51×10^{-13}.

Q. The calculator has displayed the message MEMORY FULL. What should I do?

A. You must clear a portion of memory before proceeding. (See appendix B.)

Q. Why does calculating the sine (or tangent) of π radians display a very small number instead of 0?

A. π cannot be represented *exactly* with the 12-digit precision of the calculator.

Q. Why do I get incorrect answers when I use the trigonometric functions?

A. You must make sure the calculator is using the correct angular mode (■[MODES]).

Q. What does the symbol in the display mean?

A. This is an *annunciator,* and it indicates something about the status of the calculator. See "Annunciators" in chapter 1.

Power and Batteries

The calculator is shipped with alkaline batteries. A fresh set of three alkaline batteries provides approximately a year of normal use. However, expected battery life depends on how the calculator is used; frequent, long calculations require more power than short, periodic calculations. The calculator consumes the most power while running programs or doing long calculations (like SOLVE or ∫FN). For any level of use, mercury and silver oxide batteries last about twice as long as alkaline batteries.

Use only fresh button-cell batteries. Do not use rechargeable batteries. The following batteries are recommended for use. Not all batteries are available in all countries.

Alkaline	Mercury	Silver Oxide
Panasonic LR44	Panasonic NP675	Panasonic SR44W or SP357
Eveready A76	Eveready EP675E	Eveready 357
Varta V13GA	Toshiba NR44 or MR44	Varta V357
	Radio Shack NR44 or MR44	Ray-O-Vac 357
Duracell LR44	Duracell MP675H	

Low-Power Indicator

When the low battery annunciator () comes on, you should replace the batteries as soon as possible.

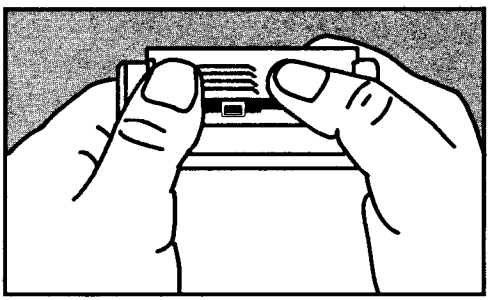
If you continue to use the calculator after the battery annunciator comes on, power can eventually drop to a level at which the display becomes dim and stored data may be affected. If this happens, the calculator requires fresh batteries before it will operate properly. If stored data has not been preserved due to extremely low power, the calculator displays MEMORY CLEAR.

Installing Batteries

Once the batteries are removed, replace them within a minute to save Continuous Memory.

To install batteries:

1. Have three fresh button-cell batteries at hand. Hold batteries by the edges. Do not touch the contacts. Wipe each battery with a clean, lint-free cloth to remove dirt and oil.
2. Make sure the calculator is *off*. **Do not press [C] again until the entire procedure for changing batteries is completed. Changing batteries with the calculator on can erase the contents of Continuous Memory.**
3. Hold the calculator as shown. To remove the battery-compartment door, press down and outward on it until it slides off (away from the center).

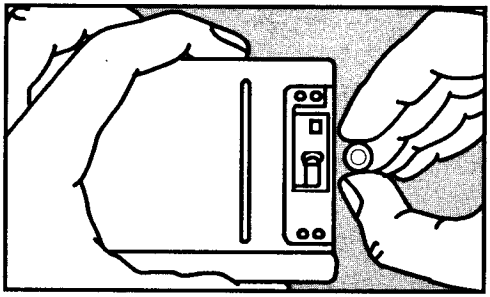


4. Turn the calculator over and shake the batteries out.

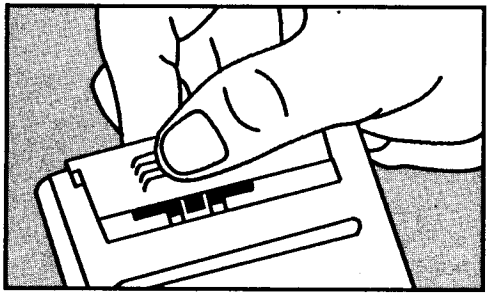
Warning **Do not mutilate, puncture, or dispose of batteries in fire. The batteries can burst or explode, releasing hazardous chemicals.**

---

5. Hold the calculator as shown and stack the batteries, one at a time, in the battery compartment. Orient the batteries according to the diagram inside the battery compartment. Be sure the raised and flat ends match the diagram.



6. Insert the tab of the battery-compartment door into the slot in the calculator case, as shown.



Now turn the calculator back on. If it does not function, check that the orientation of the batteries is correct. If the calculator still does not function, you might have taken too long to change the batteries or inadvertently turned the calculator on while the batteries were out. *Remove the batteries* again and lightly press a coin against both battery contacts in the calculator *for a few seconds.* Put the batteries back in and turn the calculator on. It should display MEMORY CLEAR.

Environmental Limits

To maintain product reliability, observe the following temperature and humidity limits:

- Operating temperature: 0° to 45°C (32° to 113°F).
- Storage temperature: −20° to 65°C (−4° to 149°F).
- Operating and storage humidity: 90% relative humidity at 40°C (104°F) maximum.

Determining if the Calculator Requires Service

Use these guidelines to determine if the calculator requires service. Then, if necessary, read "If the Calculator Requires Service" on page 249.

- **If the calculator won't turn on (nothing is visible in the display):**
  1. Attempt to reset the calculator. (Hold down the [C] key and press [LN].)
  2. If the calculator fails to respond after step 1, replace the batteries (see page 242).

  If steps 1 and 2 fail to restore calculator function, it requires service.

■ **If the calculator doesn't respond to keystrokes (nothing happens when you press any of the keys):**

1. Attempt to reset the calculator. (Hold down the [C] key and press [LN].)
2. If the calculator fails to respond after step 1, attempt to clear memory. (Hold down [C], [√x], and [Σ+] as described on page 255). This will erase all the information you've stored.
3. If the calculator fails to respond after steps 1 and 2, remove the batteries (see page 243) and lightly press a coin against both calculator battery contacts. Put the batteries back in and turn on the calculator. It should display MEMORY CLEAR.

If steps 1 through 3 fail to restore calculator function, the calculator requires service.

■ **If the calculator responds to keystrokes but you suspect that it is malfunctioning:**

1. Do the self-test (described below). If the calculator fails the self test, it requires service.
2. If the calculator passes the self-test, it is likely that you've made a mistake in operating the calculator. Try rereading portions of the manual and check "Answers to Common Questions" at the beginning of this chapter.
3. Contact the Calculator Support department. The address and phone number are listed on the inside back cover.

Confirming Calculator Operation—the Self-Test

If the display can be turned on, but it appears that the calculator is not operating properly, you can do a diagnostic self-test.

1. To start the self-test, hold down the [C] key while you press [y^x].*

* Holding down [C] as you press [1/x] starts a continuous self-test that is used at the factory. If you accidentally start this self-test, you can halt it by pressing any key.

2. Press any key eight times and watch the display as various patterns are displayed. After you've pressed the key eight times, the calculator displays the copyright message COPR. HP 1987 and then the message KBD 01.

3. Starting at the upper left corner ([√x]) and moving from left to right, press each key in the top row. Then, moving left to right, press each key in the second row, third row, etc., until you've pressed each key.
   - If you press the keys in the proper order and they are functioning properly, the calculator displays KBD followed by two-digit numbers. (The calculator is counting the keys using hexadecimal base.)
   - If you press a key out of order, or if a key isn't functioning properly, the next keystroke displays a fail message (see step 4).

4. The self-test produces one of these two results:
   - The calculator displays 32S - OK if it passed the self-test. Go to step 5.
   - The calculator displays 32S - FAIL, followed by a one-digit number, if it failed the self-test. If you received the message because you pressed a key out of order, you should reset the calculator (hold down [C] and press [LN]), and do the self-test again. If you pressed the keys in order, but got this message, repeat the self-test to verify the results. If the calculator fails again, it requires service (see page 249). Include a copy of the fail message with the calculator when you ship it for service.

5. To exit the self-test, reset the calculator (hold down [C] and press [LN]).

Limited One-Year Warranty

What Is Covered

The calculator (except for the batteries, or damage caused by the batteries) *is warranted by Hewlett-Packard against defects in materials and workmanship for one year from the date of original purchase.* If you sell your unit or give it as a gift, the warranty is automatically transferred to the new owner and remains in effect for the original one-year period. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to a Hewlett-Packard service center. (Replacement may be with a newer model of equivalent or better functionality.)

This warranty gives you specific legal rights, and you may also have other rights that vary from state to state, province to province, or country to country.

What Is Not Covered

Batteries, and damage caused by the batteries, are not covered by the Hewlett-Packard warranty. Check with the battery manufacturer about battery and battery leakage warranties.

This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than an authorized Hewlett-Packard service center.

No other express warranty is given. The repair or replacement of a product is your exclusive remedy. **ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY.** Some states, provinces, or countries do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. **IN NO EVENT SHALL HEWLETT-PACKARD COMPANY BE LIABLE FOR CONSEQUENTIAL DAMAGES.** Some states, provinces, or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Products are sold on the basis of specifications applicable at the time of manufacture. Hewlett-Packard shall have no obligation to modify or update products once sold.

Consumer Transactions in the United Kingdom

This warranty shall not apply to consumer transactions and shall not affect the statutory rights of a consumer. In relation to such transactions, the rights and obligations of Seller and Buyer shall be determined by statute.

If the Calculator Requires Service

Hewlett-Packard maintains service centers in many countries. These centers will repair a calculator or replace it (with an equivalent or newer model), whether it is under warranty or not. There is a charge for service after the warranty period. Calculators normally are serviced and reshipped within 5 working days of receipt.

Obtaining Service

- **In the United States:** Send the calculator to the Calculator Service Center listed on the inside of the back cover.
- **In Europe:** Contact your HP sales office or dealer, or HP's European headquarters for the location of the nearest service center. *Do not ship the calculator for service without first contacting a Hewlett-Packard office.*

  Hewlett-Packard S.A.
  150, Route du Nant-d'Avril
  P.O. Box CH 1217 Meyrin 2
  Geneva, Switzerland
  Telephone: (022) 82 81 11
- **In other countries:** Contact your HP sales office or dealer or write to the U.S. Calculator Service Center (listed on the inside of the back cover) for the location of other service centers. If local service is unavailable, you can ship the calculator to the U.S. Calculator Service Center for repair.

  All shipping, reimportation arrangements, and customs costs are your responsibility.

Service Charge

There is a standard repair charge for out-of-warranty service. The Calculator Service Center (listed on the inside of the back cover) can tell you how much this charge is. The full charge is subject to the customer's local sales or value-added tax wherever applicable.

Calculator products damaged by accident or misuse are not covered by the fixed service charges. In these cases, charges are individually determined based on time and material.

Shipping Instructions

If your calculator requires service, ship it to the nearest authorized service center or collection point. Be sure to:

- Include your return address and description of the problem.
- Include proof of purchase date if the warranty has not expired.
- Include a purchase order, check, or credit card number plus expiration date (Visa or MasterCard) to cover the standard repair charge. In the United States and some other countries, the serviced calculator can be returned C.O.D. if you do not pay in advance.
- Ship the calculator in adequate protective packaging to prevent damage. Such damage is not covered by the warranty, so we recommend that you insure the shipment.
- Pay the shipping charges for delivery to the Hewlett-Packard service center, whether or not the calculator is under warranty.

Warranty on Service

Service is warranted against defects in materials and workmanship for 90 days from the date of service.

Service Agreements

In the U.S., a support agreement is available for repair and service. Refer to the form that was packaged with the manual. For additional information, contact the Calculator Service Center (see the inside of the back cover).

Regulatory Information

Radio Frequency Interference

U.S.A. The HP-32S generates and uses radio frequency energy and may interfere with radio and television reception. The calculator complies with the limits for a Class B computing device as specified in Subpart J of Part 15 of FCC Rules, which provide reasonable protection against such interference in a residential installation. In the unlikely event that there is interference to radio or television reception (which can be determined by turning the calculator off and on or by removing the batteries), try:

- Reorienting the receiving antenna.
- Relocating the calculator with respect to the receiver.

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

West Germany. The HP-32S complies with VFG 1046/84, VDE 0871B, and similar non-interference standards. If you use equipment that is not authorized by Hewlett-Packard, that system configuration has to comply with the requirements of Paragraph 2 of the German Federal Gazette, Order (VFG) 1046/84, dated December 14, 1984.

B

User Memory and the Stack

This appendix covers

- The allocation and requirements of user memory,
- How to reset the calculator without affecting memory,
- How to clear (purge) all of user memory and reset the system defaults, and
- Which operations affect stack lift.

Managing Calculator Memory

The HP-32S has 390 bytes of user memory available to you for any combination of stored data (variables or program lines). SOLVE, ∫FN, and statistical calculations also require user memory. (The ∫FN operation is particularly "expensive" to run.)

All of your stored data is preserved until you explicitly clear it. The message MEMORY FULL means that there is currently not enough memory available for the operation you just attempted. You need to clear some (or all) of user memory. For instance, you can:

- Clear the contents of any or all variables (see page 50).
- Clear any or all programs (see page 85).
- Clear the statistics registers (press ■ CLEAR {Σ}).
- Clear all of user memory (press ■ CLEAR {ALL}).

Memory Requirements

Data or Operation	Amount of Memory Used
Variables	8 bytes per non-zero value. (No bytes for zero values.)
Instructions in program lines	1.5 bytes.
Numbers in program lines	Integers 0 through 99: 1.5 bytes. All other numbers: 9.5 bytes.
Statistics data	48 bytes.
SOLVE calculations	33.5 bytes.
∫FN (integration) calculations	140 bytes.

To see the total memory requirement of specific programs, press ■ [MEM] {PGM}. Press [▼] or ■ [▲] to view the entries. (For an example, see page 86.)

To manually deallocate the memory allocated for a SOLVE or ∫FN calculation that has been interrupted, press ■ [LBL/RTN] {RTN}. This deallocation is done automatically whenever you execute a program or another SOLVE or ∫FN calculation.

Resetting the Calculator

If the calculator doesn't respond to keystrokes or if it is otherwise behaving unusually, attempt to reset it. Resetting the calculator halts the current calculation and cancels program entry, digit entry, a running program, a SOLVE calculation, an ∫FN calculation, a VIEW display, or an INPUT display. Stored data usually remain intact.

To reset the calculator, hold down the [C] key and press [LN]. If you are unable to reset the calculator, try installing fresh batteries. If the calculator cannot be reset, or if it still fails to operate properly, you should attempt to clear memory using the special procedure described in the next section.

The calculator can reset itself if it is dropped or if power is interrupted.

Clearing Memory

The usual way to clear user memory is to press ■ [CLEAR] {ALL}. However, there is also a more powerful clearing procedure that resets additional information and is useful if the keyboard is not functioning properly.

If the calculator fails to respond to keystrokes, and you are unable to restore operation by resetting it or changing the batteries, try the following procedure. These keystrokes clear all of memory, reset the calculator, *and* restore all formats and modes to their original, *default* settings (shown below).

1. Press and hold down the [C] key.
2. Press and hold down [√x].
3. Press [Σ+]. (You will be pressing three keys simultaneously). When you release all three keys, the display shows MEMORY CLEAR if the operation is successful.

Default Settings

Category	Default Status
Angular mode.	Degrees.
Base mode.	Decimal.
Contrast setting.	Medium.
Decimal point.	"."
Display format.	FIX 4.
Flags.	Cleared to zero.
FN= (current function).	Null.
Program pointer (current line).	PRGM TOP.
Program memory.	Cleared.
Random-number seed.	Zero.
Stack lift.	Enabled.
Stack registers.	Cleared to zero.
Variables.	Cleared to zero.

Memory may inadvertently be cleared if the calculator is dropped or if power is interrupted.

The Status of Stack Lift

The four stack registers are always present, and the stack always has a *stack-lift status*. That is to say, the stack lift is always *enabled* or *disabled* regarding its behavior when the next number is placed in the X-register. (Refer to chapter 2, "The Automatic Memory Stack.")

Any function not in the following two lists will enable stack lift.

Disabling Operations

There are four operations that disable stack lift. A number keyed in after one of these disabling operations writes over the number currently in the X-register. The Y-, Z- and T-registers remain unchanged.

ENTER Σ+ Σ− CLx

In addition, when [C] and [◀] act like CLx, they also disable stack lift.

The INPUT function *disables* stack lift as it halts a program for prompting (so any number you then enter writes over the X-register), but it *enables* stack lift when the program resumes.

Neutral Operations

The following operations do not alter the previous status of the stack lift:

DEG,RAD, GRAD	FIX,SCI, ENG,ALL	DEC,HEX, OCT,BIN	CLVARS
PSE	SHOW	RADIX.;RADIX,	CLΣ
[OFF]	[R/S] and STOP	[▲], [▼]	[C]*, [◀]*
[MEM]{VAR}†	[MEM]{PGM}†	[GTO] [·] [·]	[GTO] [·] *label nn*
Switching binary windows	Digit entry	Errors	[PRGM] and program entry

* Except when used like CLx.

† Including all operations performed while the catalog is displayed except {VAR} [ENTER] and {PGM} [XEQ], which enable stack lift.

The Status of the LAST X Register

The following operations save x in the LAST X register:

$+, -, \times, \div$	SQRT, x^2	e^x, 10^x
LN, LOG	y^x	$1/x$
%, %CHG	$\Sigma+$, $\Sigma-$	RCL$+,-,\times,\div$*
$\hat{x}$, $\hat{y}$	SIN, COS, TAN	ASIN, ACOS, ATAN
SINH, COSH, TANH	ASINH, ACOSH, ATANH	IP, FP, RND, ABS
$y,x \rightarrow \theta,r$; $\theta,r \rightarrow y,x$	→HR, →HMS	→DEG, →RAD
Cn,r; Pn,r	$x!$	CMPLX+/−
CMPLX$+,-,\times,\div$	CMPLX e^x, LN,y^x, $1/x$	CMPLX SIN,COS,TAN

* Note that the recall-arithmetic sequence x [RCL] [+] *variable* stores a different value in the LAST X register than the sequence x [RCL] *variable* [+] does. The former stores x in LAST X; the latter stores the recalled number in LAST X.

C

More About Solving an Equation

This appendix provides information about the SOLVE operation beyond that given in chapter 7.

How SOLVE Finds a Root

SOLVE is an *iterative* operation; that is, it repetitively executes the specified function. It starts with an estimate for the unknown variable, x, and refines that estimate with each successive execution of the function, $f(x)$.*

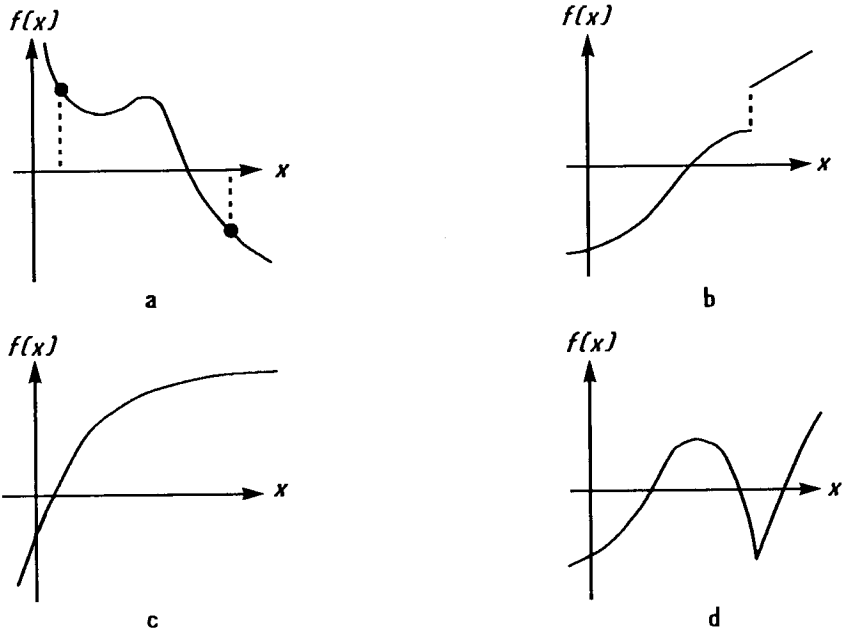
If any two successive estimates of the function $f(x)$ have opposite signs, then SOLVE presumes that the function $f(x)$ crosses the x-axis in at least one place between the two estimates. This interval is systematically narrowed until a root is found.

For SOLVE to find a root, the root has to exist within the range of numbers of the calculator, and the function must be mathematically defined where the iterative search occurs. SOLVE always finds a root, provided one exists (within the overflow bounds), if one or more of these conditions are met:

- Two estimates yield $f(x)$ values with opposite signs, and the function's graph crosses the x-axis in at least one place between those estimates (figure a, next page).

* $f(x)$ is mathematical shorthand for a function defined in terms of the unknown variable x.

- $f(x)$ always increases or always decreases as x increases (figure b, below).
- The graph of $f(x)$ is either concave everywhere or convex everywhere (figure c, below).
- If $f(x)$ has one or more local minima or maxima, each occurs singly between adjacent roots of $f(x)$ (figure d, below).



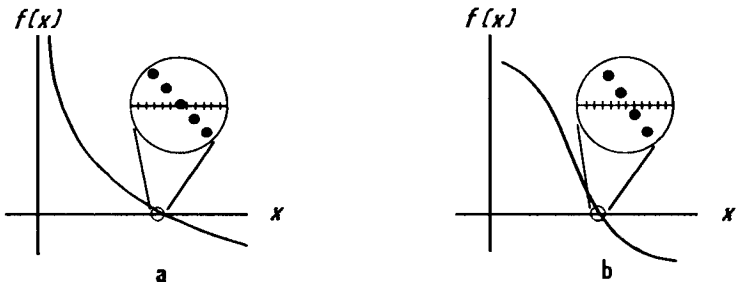
Functions Whose Roots Can Be Found

In most situations, the calculated root is an accurate estimate of the theoretical, infinitely precise root of the equation. An "ideal" solution is one for which $f(x)=0$. However, a very small non-zero value for $f(x)$ is often acceptable because it might result from approximating numbers with limited (12-digit) precision.

Interpreting Results

The SOLVE operation will produce a solution under either of these conditions:

- If it finds an estimate for which $f(x)$ equals zero (see figure a, below).
- If it finds an estimate where $f(x)$ is not equal to zero, but the calculated root is a 12-digit number adjacent to the place where the function's graph crosses the x-axis (see figure b, below). This occurs when the two final estimates are neighbors (that is, they differ by 1 in the 12th digit), and the function's value is positive for one estimate and negative for the other.* In *most* cases, $f(x)$ will be relatively close to zero.



Cases Where a Root Is Found

To obtain additional information about the result, press [R↓] to see the previous estimate of the root (x), which was left in the Y-register. Press [R↓] again to see the value of $f(x)$, which was left in the Z-register. If $f(x)$ equals zero or is relatively small, it is very likely that a solution has been found. However, if $f(x)$ is relatively large, you must use caution in interpreting the results.

* Or they are $(0, 10^{-499})$ or $(0, -10^{-499})$.

Example: An Equation With One Root. Find the root of the equation:

$$-2x^3 + 4x^2 - 6x + 8 = 0,$$

which, using Horner's method (chapter 5), simplifies to

$$x(x(-2x + 4) - 6) + 8 = 0.$$

Enter the function as the program:

```
A01 LBL A
A02 -2
A03 RCL× X
A04 4
A05 +
A06 RCL× X
A07 6
A08 -
A09 RCL× X
A10 8
A11 +
A12 RTN
```

Keys:	Display:	Description:
■ SOLVE/∫ {FN}A 0 STO X 10 ■ SOLVE/∫ {SOLVE}X	X=1.6506	Calculates x using guesses 0 and 10.
R↓	1.6506	Final two estimates are the same to four decimal places.
R↓	-1.0000E-11	$f(x)$ is *very* small, so the approximation is a good root.

Example: An Equation With Two Roots. Find the two roots of the parabolic equation:

$$x^2 + x - 6 = 0.$$

Enter the function as the program:

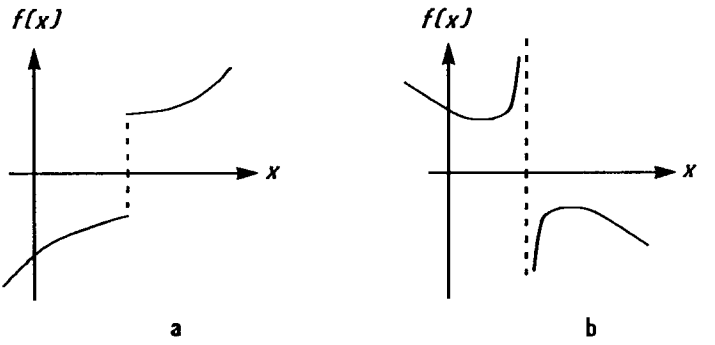
```
D01 LBL D
D02 RCL X
D03 x²
D04 RCL+ X
D05 6
D06 -
D07 RTN
```

Keys:	Display:	Description:
■ SOLVE/∫ {FN} D 0 STO X 10 ■ SOLVE/∫ {SOLVE} X	X=2.0000	Calculates the positive root using guesses 0 and 10.
R↓	2.0000	Final two estimates are the same.
R↓ ■ SHOW	0.00000000000	$f(x) = 0$.
0 STO X 10 +/- ■ SOLVE/∫ {SOLVE}X	X=-3.0000	Calculates the negative root using guesses 0 and −10.
R↓ R↓ ■ SHOW	0.00000000000	$f(x) = 0$.

Certain cases require special consideration:

- If the function's graph has a discontinuity that crosses the x-axis, then the SOLVE operation returns a value adjacent to the discontinuity (see figure a, next page). In this case, $f(x)$ may be relatively large.

- Values of $f(x)$ may be approaching infinity at the location where the graph changes sign (see figure b, below). This situation is called a *pole*. Since the SOLVE operation determines that there is a sign change between two neighboring values of x, it returns the possible root. However, the value for $f(x)$ will be relatively large. If the pole occurs at a value of x that is exactly represented with 12 digits, then that value would cause the calculation to halt with an error message.



a b

Special Cases: A Discontinuity and a Pole

Example: A Discontinuous Function. Find the root of the equation:

$$\text{IP}(x) - 1.5 = 0.$$

Enter the function as the program:

```
E01 LBL E
E02 RCL X
E03 IP
E04 1.5
E05 -
R06 RTN
```

Keys:	Display:	Description:
■ SOLVE/∫ {FN}E 0 STO X 5 ■ SOLVE/∫ {SOLVE}X	X=2.0000	Finds a root with guesses 0 and 5.
■ SHOW	1.99999999999	Shows root to 11 decimal places.
R↓ ■ SHOW	2.00000000000	The previous estimate is slightly bigger.
R↓	-0.5000	*f(x)* is relatively large.

Note the difference between the last two estimates, as well as the relatively large value for *f(x)*. The problem is that there is no value of x for which *f(x)* equals zero. However, at $x = 1.99999999999$, there is a neighboring value of x that yields an opposite sign for *f(x)*.

Example: A Pole. Find the root of the equation

$$\frac{x}{x^2 - 6} - 1 = 0.$$

As x approaches $\sqrt{6}$, f(x) becomes a very large positive or negative number.

Enter the function as the program:

```
F01 LBL F
F02 RCL X
F03 x²
F04 6
F05 -
F06 RCL X
F07 x<>y
F08 ÷
F09 1
F10 -
F11 RTN
```

Note that you can shorten the program by deleting lines F06–F07 and adding a second RCL X instruction after line F02.

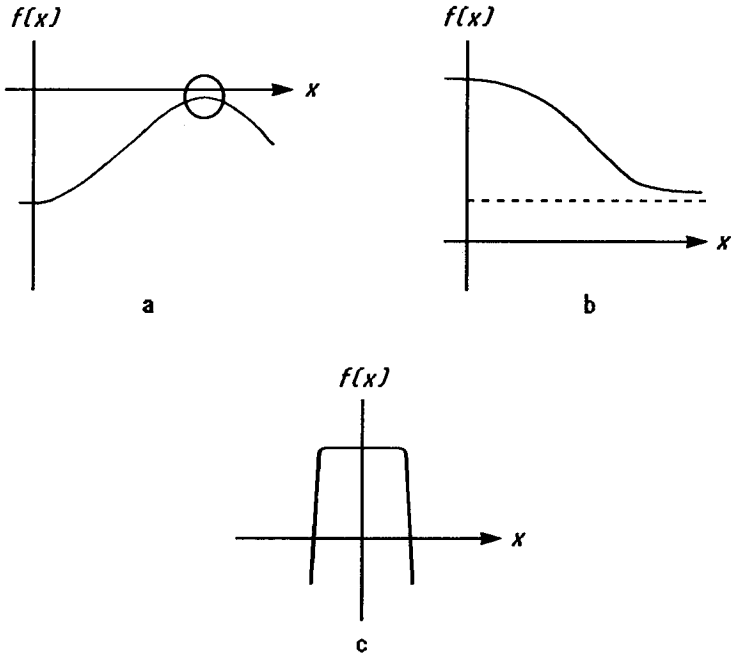
Keys:	**Display:**	**Description:**
[SOLVE/∫] {FN}F 2.3 [STO] X 2.7 [SOLVE/∫] {SOLVE}X	X=2.4495	Calculates the root using guesses that bracket $\sqrt{6}$.
[R↓][R↓]	81,649,658,092.0	$f(x)$ is relatively large.

There is a pole between the final estimates. The initial guesses yielded opposite signs for $f(x)$, and the interval between successive estimates was narrowed until two neighbors were found. Unfortunately, these neighbors made $f(x)$ approach a pole instead of the x-axis. The function *does* have roots at -2 and 3, which can be found by entering better guesses.

When SOLVE Cannot Find a Root

Sometimes SOLVE fails to find a root. The following conditions cause the message NO ROOT FND:

- The search terminates near a local minimum or maximum (see figure a, below). If the ending value of *f(x)* (stored in the Z-register) is relatively close to zero, it is possible that a root has been found; the number stored in the unknown variable might be a 12-digit number very close to a theoretical root.
- The search halts because SOLVE is working on a horizontal asymptote—an area where *f(x)* is essentially constant for a wide range of *x* (see figure b, below). The ending value of *f(x)* is the value of the potential asymptote.
- The search is concentrated in a local "flat" region of the function (see figure c, below). The ending value of *f(x)* is the value of the function in this region.



Cases Where a Root Is Not Found

The SOLVE operation returns a math error if an estimate produces an operation that is not allowed—for example, division by zero, a square root of a negative number, or a logarithm of zero. Keep in mind that SOLVE can generate estimates over a wide range. You can sometimes avoid math errors by using good guesses. If a math error occurs, press [RCL] *unknown variable* (or [■] [VIEW] *variable*) to see the value that produced the error.

Example: A Relative Minimum. Calculate the root of this parabolic equation:

$$x^2 - 6x + 13 = 0.$$

It has a minimum at $x = 3$.

Enter the function as the program:

```
G01 LBL G
G02 RCL X
G03 x²
G04 6
G05 RCL× X
G06 -
G07 13
G08 +
G09 RTN
```

Keys:	Display:	Description:
[■] [SOLVE/∫] {FN}G 0 [STO] X 10 [■] [SOLVE/∫] {SOLVE}X	NO ROOT FND	Search fails with guesses 0 and 10.
[←] [■] [SHOW]	3.00000010001	Displays the final estimate of x.
[R↓] [■] [SHOW]	3.00000468443	Previous estimate was not the same.
[R↓]	4.0000	Final value for $f(x)$ is relatively large.

Example: An Asymptote. Find the root of the equation

$$10 - \frac{1}{x} = 0.$$

Enter the function as the program:

```
H01 LBL H
H02 10
H03 RCL X
H04 1/x
H05 -
H06 RTN
```

Keys:	Display:	Description:
■ SOLVE/∫ {FN}H .005 STO X 5 ■ SOLVE/∫ {SOLVE}X	X=0.1000	Solves for x using guesses 0.005 and 5.
R↓	0.1000	Previous estimate is the same.
R↓ ■ SHOW	0.00000000000	$f(x) = 0$.

Watch what happens when you use negative values for guesses:

Keys:	Display:	Description:
1 +/- STO X 2 +/- ■ SOLVE/∫ {SOLVE}X	NO ROOT FND	No root found for $f(x)$.
◄	-46,666,666,692.1	Displays last estimate of x.
R↓	-5.7750E15	Previous estimate was much larger in magnitude.
R↓	10.0000	$f(x)$ for last estimate is rather large.

It's apparent from inspecting the equation that if x is a negative number, the smallest that $f(x)$ can be is 10. $f(x)$ *approaches* 10 as x becomes a negative number of large magnitude.

Example: A Math Error. Find the root of the equation:

$$\sqrt{[x \div (x + 0.3)]} - 0.5 = 0.$$

Enter the function as the program:

```
I01 LBL I
I02 RCL X
I03 0.3
I04 RCL+ X
I05 ÷
I06 SQRT
I07 0.5
I08 -
I09 RTN
```

First attempt to find a positive root.

Keys:	Display:	Description:
■ SOLVE/∫ {FN} I		Calculates the root using guesses 0 and 10.
0 STO X 10		
■ SOLVE/∫ {SOLVE}		
X	X=0.1000	

Now attempt to find a negative root by entering guesses 0 and −10. Notice that the function is undefined for values of x between 0 and −0.3 since those values produce a positive denominator but a negative numerator, causing a negative square root.

0 STO X 10 +/−		Math error.
■ SOLVE/∫ {SOLVE}		
X	SQRT(NEG)	
■ VIEW X	X=-0.1308	Displays the final estimate of x.

Example: A Local "Flat" Region. Find the root of the function

$$f(x) = \begin{cases} x + 2 \text{ if } x < -1 \\ 1 \text{ for } -1 \leq x \leq 1 \\ -x + 2 \text{ if } x > 1 \end{cases} \quad \text{(a local flat region)}$$

Enter the function as the program:

```
J01 LBL J
J02 1
J03 ENTER *
J04 2
J05 RCL+ X
J06 x<y?
J07 RTN
J08 4
J09 -
J10 +/-
J11 x>y?
J12 R↓
J13 RTN
```

Solve for X using initial guesses of 10^{-8} and -10^{-8}.

Keys:	Display:	Description:
[SOLVE/∫] {FN} J [E] 8 [+/-] [STO] X 1 [+/-] [E] 8 [+/-] [SOLVE/∫] {SOLVE} X	NO ROOT FND	No root found using very small guesses near zero (thereby restricting the search to the flat region of the function).
[◆] [R↓] [R↓]	1.0000E-8 0.0025 1.0000	The last two estimates are far apart, and the final value of $f(x)$ is large.

If you use larger guesses, then SOLVE can find the roots, which are outside the flat region (at $x=2$ and $x=-2$).

* You can subsequently delete line J03 to save memory.

Round-Off Error and "Underflow"

Round-off Error. The limited (12-digit) precision of the calculator can cause errors due to rounding off, which adversely affect the iterative solutions of SOLVE and integration. For example,

$$[(|x| + 1) + 10^{15}]^2 - 10^{30} = 0$$

has no roots because $f(x)$ is always greater than zero. However, given initial guesses of 1 and 2, SOLVE returns the answer 1.0000 due to round-off error.

Round-off error can also cause SOLVE to fail to find a root. The equation

$$|x^2 - 7| = 0$$

has a root at $\sqrt{7}$. However, no 12-digit number *exactly* equals $\sqrt{7}$, so the calculator can never make the function equal to zero. Furthermore, the function never changes sign. SOLVE returns the message NO ROOT FND. However, the final estimate of x (press ◀ to see it) is the best possible 12-digit approximation of the root when the routine quits.

"Underflow." *Underflow* occurs when the magnitude of a number is smaller than the calculator can represent, so it substitutes zero. This can affect SOLVE results. For example, consider the equation

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

whose root is infinite in value. Because of underflow, SOLVE returns a very large value as a root. (The calculator cannot represent infinity, anyway.)

D

More About Integration

This appendix provides information about integration beyond that given in chapter 8.

How the Integral Is Evaluated

The algorithm used by the integration operation, ∫FN d x, calculates the integral of a function $f(x)$ by computing a weighted average of the function's values at many values of x (known as sample points) within the interval of integration. The accuracy of the result of any such sampling process depends on the number of sample points considered: generally, the more sample points, the greater the accuracy. If $f(x)$ could be evaluated at an infinite number of sample points, the algorithm could—neglecting the limitation imposed by the inaccuracy in the calculated function $f(x)$—always provide an exact answer.

Evaluating the function at an infinite number of sample points would take forever. However, this is not necessary since the maximum accuracy of the calculated integral is limited by the accuracy of the calculated function values. Using only a finite number of sample points, the algorithm can calculate an integral that is as accurate as is justified considering the inherent uncertainty in $f(x)$.

The integration algorithm at first considers only a few sample points, yielding relatively inaccurate approximations. If these approximations are not yet as accurate as the accuracy of $f(x)$ would permit, the algorithm is iterated (repeated) with a larger number of sample points. These iterations continue, using about twice as many sample points each time, until the resulting approximation is as accurate as is justified considering the inherent uncertainty in $f(x)$.

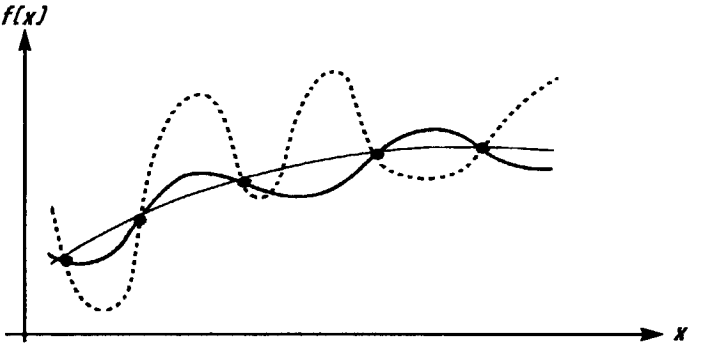
As explained in chapter 8, the uncertainty of the final approximation is a number derived from the display format, which specifies the uncertainty for the function. At the end of each iteration, the algorithm compares the approximation calculated during that iteration with the approximations calculated during two previous iterations. If the difference between any of these three approximations and the other two is less than the uncertainty tolerable in the final approximation, the calculations ends, leaving the current approximation in the X-register and its uncertainty in the Y-register.

It is extremely unlikely that the errors in each of three successive approximations—that is, the differences between the actual integral and the approximations—would all be larger than the disparity among the approximations themselves. Consequently, the error in the final approximation will be less than its uncertainty (provided that $f(x)$ does not vary rapidly). Although we can't know the error in the final approximation, the error is extremely unlikely to exceed the displayed uncertainty of the approximation. In other words, the uncertainty estimate in the Y-register is an almost certain "upper bound" on the difference between the approximation and the actual integral.

Conditions That Could Cause Incorrect Results

Although the integration algorithm in the HP–32S is one of the best available, in certain situations it—like all other algorithms for numerical integration—might give you an incorrect answer. *The possibility of this occurring is extremely remote.* The algorithm has been designed to give accurate results with almost any *smooth* function. Only for functions that exhibit *extremely* erratic behavior is there any substantial risk of obtaining an inaccurate answer. Such functions rarely occur in problems related to actual physical situations; when they do, they usually can be recognized and dealt with in a straightforward manner.

Unfortunately, since all that the algorithm knows about $f(x)$ are its values at the sample points, it cannot distinguish between $f(x)$ and any other function that agrees with $f(x)$ at all the sample points. This situation is depicted below, showing (over a portion of the interval of integration) three functions whose graphs include the many sample points in common.



With this number of sample points, the algorithm will calculate the same approximation for the integral of any of the functions shown. The actual integrals of the functions shown with solid and dashed lines are about the same, so the approximation will be fairly accurate if $f(x)$ is one of these functions. However, the actual integral of the function shown with a dotted line is quite different from those of the others, so the current approximation will be rather inaccurate if $f(x)$ is this function.

The algorithm comes to know the general behavior of the function by sampling the function at more and more points. If a fluctuation of the function in one region is not unlike the behavior over the rest of the interval of integration, at some iteration the algorithm will likely detect the fluctuation. When this happens, the number of sample points is increased until successive iterations yield approximations that take into account the presence of the most rapid, *but characteristic,* fluctuations.

For example, consider the approximation of

$$\int_0^{\infty} xe^{-x}dx.$$

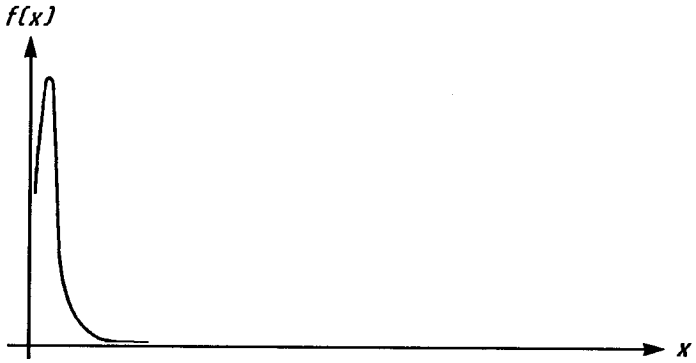
Since you're evaluating this integral numerically, you might think that you should represent the upper limit of integration as 10^{499}, which is virtually the largest number you can key into the calculator. Try it and see what happens. Enter this program that evaluates the function $f(x) = xe^{-x}$.

```
F01 LBL F
F02 RCL X
F03 +/-
F04 e^x
F05 RCL× X
F06 RTN
```

Set the display format to SCI 3, specify the lower and upper limits of integration as zero and 10^{499}, then start the integration.

Keys:	Display:	Description:
■ DISP {SC} 3 0 ENTER E 499	1E499_	Specifies accuracy level and limits of integration.
■ SOLVE/∫ {FN} F ■ SOLVE/∫ {∫FN} X	∫=0.000E0	Approximation of integral.

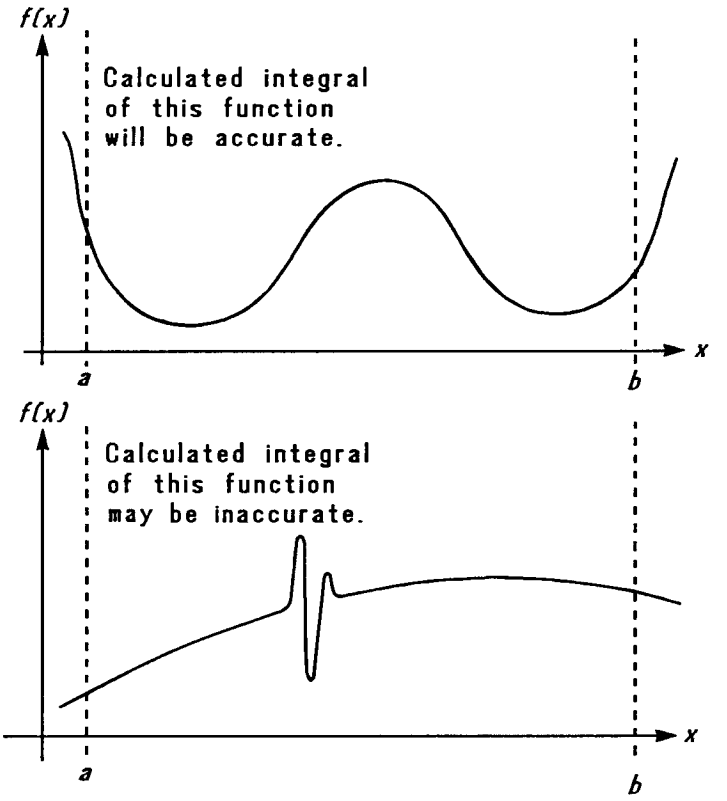
The answer returned by the calculator is clearly incorrect, since the actual integral of $f(x) = xe^{-x}$ from zero to ∞ is exactly 1. But the problem is *not* that ∞ was represented by 10^{499}, since the actual integral of this function from zero to 10^{499} is very close to 1. The reason for the incorrect answer becomes apparent from the graph of $f(x)$ over the interval of integration:



The graph is a spike very close to the origin. Because no sample point happened to discover the spike, the algorithm assumed that $f(x)$ was identically equal to zero throughout the interval of integration. Even if you increased the number of sample points by calculating the integral in SCI 11 or ALL format, none of the additional sample points would discover the spike when this particular function is integrated over this particular interval. (For better approaches to problems such as this, see the next topic, "Conditions That Prolong Calculation Time.")

Fortunately, functions exhibiting such aberrations (a fluctuation that is uncharacteristic of the behavior of the function elsewhere) are unusual enough that you are unlikely to have to integrate one unknowingly. A function that could lead to incorrect results can be identified in simple terms by how rapidly it and its low-order derivatives vary across the interval of integration. Basically, the more rapid the variation in the function or its derivatives, and the lower the order of such rapidly varying derivatives, the less quickly will the calculation finish, and the less reliable will be the resulting approximation.

Note that the rapidity of variation in the function (or its low-order derivatives) must be determined with respect to the width of the interval of integration. With a given number of sample points, a function $f(x)$ that has three fluctuations can be better characterized by its samples when these variations are spread out over most of the interval of integration than if they are confined to only a small fraction of the interval. (These two situations are shown in the following two illustrations.) Considering the variations or fluctuation as a type of oscillation in the function, the criterion of interest is the ratio of the period of the oscillations to the width of the interval of integration: the larger this ratio, the more quickly the calculation will finish, and the more reliable will be the resulting approximation.



In many cases you will be familiar enough with the function you want to integrate that you will know whether the function has any quick wiggles relative to the interval of integration. If you're not familiar with the function, and you suspect that it may cause problems, you can quickly plot a few points by evaluating the function using the subroutine you wrote for that purpose.

If, for any reason, after obtaining an approximation to an integral, you suspect its validity, there's a simple procedure to verify it: subdivide the interval of integration into two or more adjacent subintervals, integrate the function over each subinterval, then add the resulting approximations. This causes the function to be sampled at a brand new set of sample points, thereby more likely revealing any previously hidden spikes. If the initial approximation was valid, it will equal the sum of the approximation over the subintervals.

Conditions That Prolong Calculation Time

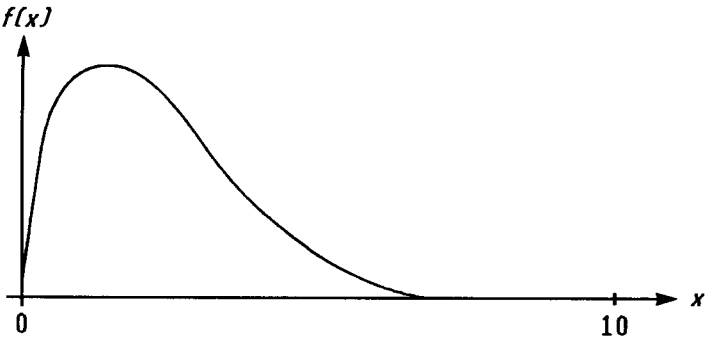
In the preceding example, the algorithm gave an incorrect answer because it never detected the spike in the function. This happened because the variation in the function was too quick relative to the width of the interval of integration. If the width of the interval were smaller, you would get the correct answer; but it would take a very long time if the interval were still too wide.

Consider an integral where the interval of integration is wide enough to require excessive calculation time, but not so wide that it would be calculated incorrectly. Note that because $f(x) = xe^{-x}$ approaches zero very quickly as x approaches ∞, the contribution to the integral of the function at large values of x is negligible. Therefore, you can evaluate the integral by replacing ∞, the upper limit of integration, by a number not so large as 10^{499}—say 10^3.

Re-run the previous integration problem with this new limit of integration. If you have not run any other integrations in the meantime, you do not have to re-specify FN= F.

Keys:	Display:	Description:
0 [ENTER] [E] 3	1E3_	New upper limit.
■[SOLVE/∫] {∫FN} X	∫=1.000E0	Integral. (The calculation takes a while.)
[x≷y]	1.824E-4	Uncertainty of approximation.

This is the correct answer, but it took a very long time. To understand why, compare the graph of the function between $x = 0$ and $x = 10^3$, which looks about the same as that shown on page 276, with the graph of the function between $x = 0$ and $x = 10$:



You can see that this function is "interesting" only at small values of x. At greater values of x, the function is not interesting, since it decreases smoothly and gradually in a predictable manner.

The algorithm samples the function with higher densities of sample points until the disparity between successive approximations becomes sufficiently small. For a narrow interval in an area where the function is interesting, it takes less time to reach this critical density.

To achieve the same density of sample points, the total number of sample points required over the larger interval is much greater than the number required over the smaller interval. Consequently, several more iterations are required over the larger interval to achieve an approximation with the same accuracy, and therefore calculating the integral requires considerably more time.

Because the calculation time depends on how soon a certain density of sample points is achieved in the region where the function is interesting, the calculation of the integral of any function will be prolonged if the interval of integration includes mostly regions where the function is not interesting. Fortunately, if you must calculate such an integral, you can modify the problem so that the calculation time is considerably reduced. Two such techniques are subdividing the interval of integration and transformation of variables. These methods enable you to change the function or the limits of integration so that the integrand is better behaved over the interval(s) of integration.

Messages

The calculator responds to certain conditions or keystrokes by displaying a message. The ⚠ symbol comes on to call your attention to the message. For significant conditions, the message remains until you clear it. Pressing [C] or [◀] clears the message; pressing any other key clears the message *and* executes that key's function.

∫FN ACTIVE
A running program attempted to select a program label (FN=*label*) while an integration calculation was running.

∫(∫FN)
A running program attempted to calculate an integral (∫FN d *variable*) while another integration calculation was running.

∫(SOLVE)
A running program attempted a SOLVE operation while an integration calculation was running.

ALL VARS=0
The catalog of variables ([■] [MEM] {VAR}) indicates no values stored.

CALCULATING
The calculator is executing a function that might take a while.

DIVIDE BY 0
Attempted to divide by zero. (Includes [%CHG] if Y-register contains zero.)

DUPLICAT. LBL
Attempted to record a program label that already exists for another program routine.

INTEGRATING
The calculator is calculating an integral. This *might* take a while.

INVALID DATA
Data error:

- Attempted to calculate combinations or permutations with $r>n$, with non-integer r or n, or with $n \geqslant 10^{12}$.
- Attempted to use a trigonometric or hyperbolic function with an illegal argument: [TAN] with x an odd multiple of 90°; [ACOS] or [ASIN] with $x < -1$ or $x > 1$; [HYP][ATAN] with $x \leqslant -1$ or $x \geqslant 1$; [HYP][ACOS] with $x < 1$.

INVALID x!
Attempted a factorial or gamma operation with x as a negative integer.

INVALID y^x
Exponentiation error:

- Attempted to raise 0 to the 0th or to a negative power.
- Attempted to raise a negative number to a non-integer power.
- Attempted to raise the complex number $(0 + i0)$ to a number with a negative real part.

INVALID (i)
Attempted an operation with an indirect address, but the number in the index register is invalid ($|i| \geqslant 27$ or $0 \leqslant |i| < 1$).

LOG(0)
Attempted to take a logarithm of zero or $(0 + i0)$.

LOG(NEG)
Attempted to take a logarithm of a negative number.

MEMORY CLEAR
All of user memory has been erased (see page 255).

MEMORY FULL
The calculator has insufficient memory available to do the operation. See appendix B.

NONEXISTENT
Attempted to refer to a nonexistent program label (or line number) with [GTO], [GTO][.], [XEQ], or {FN}. Note that the error NONEXISTENT can mean either (1) you explicitly (from the keyboard) called a program label that does not exist; or (2) the program that you called referred to *another* label, which does not exist.

NO LABELS
The catalog of programs ([■][MEM] {PGM}) indicates no program labels stored.

NO ROOT FND
SOLVE cannot find the root of the equation using the current initial guesses (see page 120 and page 261). A SOLVE operation executed in a program does not produce this error; the same condition causes it instead to skip the next program line (the line following the instruction SOLVE *variable*).

NO STAT DATA
Attempted to do a statistics calculation with no statistics data stored.

OVERFLOW
Warning (displayed momentarily); the magnitude of a result is too large for the calculator to handle. The calculator returns ±9.99999999999E499 in the current display format. (See "Range of Numbers and Overflow" on page 24.) This condition sets flag 6. If flag 5 is set, then overflow has the added effect of halting a running program and leaving the message in the display until you press a key.

PRGM TOP
Indicates the "top" of program memory. The memory scheme is circular, so PRGM TOP is also the line after the last line in program memory.

RUNNING
The calculator is running a program (other than a SOLVE or ∫FN routine).

SELECT FN
Attempted to execute SOLVE *variable* or ∫FN d *variable* without a selected program label. This can happen only the first time that you use SOLVE or ∫FN after the message MEMORY CLEAR, or it can happen if the current label no longer exists.

SOLVE ACTIVE
A running program attempted to select a program label (FN=*label*) while a SOLVE operation was running.

SOLVE(SOLVE)
A running program attempted a SOLVE operation while another SOLVE operation was running.

SOLVE(∫FN)
A running program attempted to calculate an integral while a SOLVE operation was running.

SOLVING
The calculator is solving an equation for its root. This *might* take a while.

SQRT(NEG)
Attempted to calculate the square root of a negative number.

STAT ERROR
Statistics error:

- Attempted to calculate s_x s_y $\hat{x}$, $\hat{y}$, m, r, or b with $n = 1$.
- Attempted to calculate r, $\hat{x}$, or $\bar{x}w$ with x-data only (all y-values equal to zero).
- Attempted to calculate $\hat{x}$, $\hat{y}$, r, m, or b with all x-values equal.
- Attempted to do a statistics calculation after [Σ−] has reduced n to zero.

TOO BIG

The magnitude of the number is too large to be converted to HEX, OCT, or BIN base. The number must be in the range $-34{,}359{,}738{,}368 \leqslant n \leqslant 34{,}359{,}738{,}367$.

XEQ OVERFLOW

A running program attempted an eighth nested XEQ *label*. (Up to seven subroutines can be nested.) Since SOLVE and ∫FN each use a level, they can also generate this error.

Function Index

This section is a quick reference for all functions and operations and their formulas, where appropriate. The listing is in alphabetical order by the function's name. This name is the one used in program lines. For example, the function named FIX *n* is executed as ■ [DISP] {FX}*n*.

Those functions that are not programmable have their names in key boxes, such as [◆].

Non-letter characters and Greek letters are alphabetized before all the letters; function names preceded by arrows (e.g. →DEG) are alphabetized as if the arrow were not there.

Function Name	Keys and Description	Page
+/−	[+/−] Changes the sign of a number.	21
+	[+] *Addition.* Returns $y + x$.	25
−	[−] *Subtraction.* Returns $y - x$.	25
×	[×] *Multiplication.* Returns $y \times x$.	25
÷	[÷] *Division.* Returns $y \div x$.	25
[◆]	Deletes the last digit keyed in; clears *x*; clears a menu; deletes a program step.	16, 19, 32, 83
■[▲]	Displays previous entry in catalog; moves program pointer to previous step.	33, 76

Function Name	Keys and Description	Page
▼	Displays next entry in catalog; moves program pointer to next step (during program entry); executes the current program line (not during program entry).	33, 76
1/x	[1/x] *Reciprocal.*	24
10^x	[■] [10^x] *Common exponential.* Returns 10 raised to the *x* power.	55
%	[■] [%] *Percent.* Returns $(y \times x) \div 100$.	59
%CHG	[■] [%CHG] *Percent change.* Returns $(x - y)(100 \div y)$.	59
π	[■] [π] Returns the approximation 3.14159265359.	56
Σ+	[Σ+] Accumulates (*y*, *x*) into statistics registers.	154
Σ−	[■] [Σ−] Removes (*y*, *x*) from statistics registers.	155
Σ*x*	[■] [STAT] {Σ} {*x*} Returns the sum of *x*-values.	161
Σx^2	[■] [STAT] {Σ} {x^2} Returns the sum of squares of *x*-values.	162
Σ*xy*	[■] [STAT] {Σ} {*xy*} Returns the sum of products of *x*- and *y*-values.	162
Σ*y*	[■] [STAT] {Σ} {*y*} Returns the sum of *y*-values.	161
Σy^2	[■] [STAT] {Σ} {y^2} Returns the sum of squares of *y*-values.	162

Function Name	Keys and Description	Page
θ,r→y,x	[■] [P↔RECT] {θ,r→y,x} *Polar to rectangular.* Converts (r, θ) to (x, y).	61
∫FN d *variable*	[■] [SOLVE/∫] {∫FN} *variable* *Integrates* the current function with respect to the *variable*, using lower limit in Y-register and upper limit in X-register.	127
ABS	[■] [PARTS] {ABS} *Absolute value.* Returns $\|x\|$.	67
ACOS	[■] [ACOS] *Arc cosine.* Returns $\cos^{-1} x$.	57
ACOSH	[■] [HYP] [■] [ACOS] *Hyperbolic arc cosine.* Returns $\cosh^{-1} x$.	59
ALL	[■] [DISP] {ALL} Selects display of all significant digits.	30
ASIN	[■] [ASIN] *Arc sine.* Returns $\sin^{-1} x$.	57
ASINH	[■] [HYP] [■] [ASIN] *Hyperbolic arc sine.* Returns $\sinh^{-1} x$.	59
ATAN	[■] [ATAN] *Arc tangent.* Returns $\tan^{-1} x$.	57
ATANH	[■] [HYP] [■] [ATAN] *Hyperbolic arc tangent.* Returns $\tanh^{-1} x$.	59
b	[■] [STAT] {L.R.} {b} Returns the *y-intercept* of the regression line: $\bar{y} - m\bar{x}$.	159
[■] [BASE]	Displays the menu for base conversions.	144
BIN	[■] [BASE] {BN} Selects Binary (base 2) mode.	144
[C]	Turns on calculator; clears x; clears messages and prompts; cancels menus; cancels catalogs; cancels program entry; halts a running program.	14, 16, 19, 32, 36, 73

Function Name	Keys and Description	Page
CF *n*	■ [FLAGS] {CF}*n* *Clears flag n* ($0 \leq n \leq 6$).	98
■ [CLEAR]	Displays the menu to clear numbers or parts of memory; or clears the indicated variable or program from a MEM catalog.	16, 33
■ [CLEAR] {ALL}	Clears all stored data and programs.	34
■ [CLEAR] {PGM}	Clears all programs.	86
CLΣ	■ [CLEAR] {Σ} Clears statistics registers.	154
CLVARS	■ [CLEAR] {VARS} Clears all variables to zero.	50
CLx	■ [CLEAR] {x} Clears *x* to zero.	36, 40, 73
■ [CMPLX]	Displays the CMPLX_ prefix for complex functions.	139
CMPLX+/−	■ [CMPLX] [+/−] *Complex change sign.* Returns $-(z_x + iz_y)$.	139
CMPLX+	■ [CMPLX] [+] *Complex addition.* Returns $(z_{1x} + iz_{1y}) + (z_{2x} + iz_{2y})$.	140
CMPLX−	■ [CMPLX] [−] *Complex subtraction.* Returns $(z_{1x} + iz_{1y}) - (z_{2x} + iz_{2y})$.	140
CMPLX×	■ [CMPLX] [×] *Complex multiplication.* Returns $(z_{1x} + iz_{1y}) \times (z_{2x} + iz_{2y})$.	140

Function Name	Keys and Description	Page
CMPLX÷	■ [CMPLX] [÷] *Complex division.* Returns $(z_{1x} + iz_{1y}) \div (z_{2x} + iz_{2y})$.	140
CMPLX1/x	■ [CMPLX] [1/x] *Complex reciprocal.* Returns $1/(z_x + iz_y)$.	139
CMPLXCOS	■ [CMPLX] [COS] *Complex cosine.* Returns cos $(z_x + iz_y)$.	139
CMPLX e^x	■ [CMPLX] [e^x] *Complex natural exponential.* Returns $e^{z_x + iz_y}$.	139
CMPLXLN	■ [CMPLX] [LN] *Complex natural log.* Returns $\log_e (z_x + iz_y)$.	139
CMPLXSIN	■ [CMPLX] [SIN] *Complex sine.* Returns sin $(z_x + iz_y)$.	139
CMPLXTAN	■ [CMPLX] [TAN] *Complex tangent.* Returns tan $(z_x + iz_y)$.	139
CMPLX y^x	■ [CMPLX] [y^x] *Complex power.* Returns $(z_{1x} + iz_{1y})^{(z_{2x} + iz_{2y})}$.	140
Cn,r	■ [PROB] {Cn,r} *Combinations* of *n* items taken *r* at a time. Returns $n! \div (r!(n - r)!)$.	65
COS	[COS] *Cosine.* Returns cos *x*.	57
COSH	■ [HYP] [COS] *Hyperbolic cosine.* Returns cosh *x*.	59
DEC	■ [BASE] {DEC} Selects Decimal mode.	144
→DEG	■ [D↔RAD] {→DEG} *Radians to degrees.* Returns $(360/2\pi)x$.	64

Function Name	Keys and Description	Page
DEG	[MODE] {DG} Selects Degrees angular mode.	57
[DISP]	Displays the menu to adjust the display format.	30
[D↔RAD]	Displays the menu to convert between degrees and radians.	64
DSE *variable*	[LOOP] {DSE} *variable* *Decrement, Skip if Equal* or less. For control number *ccccccc.fffii* stored in a variable, subtracts *ii* (incremental value) from *ccccccc* (counter value) and, if the result ⩽ *fff* (final value), skips the next program line.	101
[E]	Begins entry of exponents and adds "E" to the number being entered. Indicates that a power of ten follows.	22
e^x	[e^x] *Natural exponential.* Returns *e* raised to the *x* power.	55
ENG *n*	[DISP] {EN} *n* Selects Engineering display with *n* digits following the first digit. $0 \leqslant n \leqslant 11$.	30
ENTER	[ENTER] Separates two numbers keyed in sequentially; copies *x* into the Y-register, lifts *y* into the Z-register, lifts *z* into the T-register, and loses *t*.	23, 39
FIX *n*	[DISP] {FX} *n* Selects Fixed display with *n* decimal places. $0 \leqslant n \leqslant 11$.	30
[FLAGS]	Displays the menu to set, clear, and test flags.	98
FN= *label*	[SOLVE/∫] {FN} *label* Selects the *label*ed program as the current function (used by SOLVE and ∫FN).	111, 127
FP	[PARTS] {FP} *Fractional part* of *x*.	67

Function Name	Keys and Description	Page
FS? *n*	[■] [FLAGS] {FS?} *n* If flag *n* ($0 \leq n \leq 6$) is set, executes the next program line; if flag *n* is clear, skips the next program line.	98
GRAD	[■] [MODES] {GR} Sets Grads angular mode.	57
GTO *label*	[■] [GTO] *label* Sets the program pointer to the program *label* in program memory.	93, 100
[■] [GTO] [.] *label nn*	Sets the program pointer to the program line *label nn*.	94
[■] [GTO] [.] [.]	Sets the program pointer to PRGM TOP.	94
HEX	[■] [BASE] {HX} Selects Hexadecimal (base 16) mode.	144
[■] [HYP]	Displays the HYP_ prefix for hyperbolic functions.	59
[■] [H↔HMS]	Displays the menu to convert between fractional hours and hours-minutes-seconds.	63
→HMS	[■] [H↔HMS] {→HMS} *Hours to hours, minutes, seconds.* Converts *x* from a decimal fraction to minutes-seconds format.	64
→HR	[■] [H↔HMS] {→HR} *Hours, minutes, seconds to hours.* Converts *x* from minutes-seconds format to a decimal fraction.	64
[(i)]	The indirect parameter. Addresses (indirectly) the variable or label whose letter corresponds to the numeric value in the variable *i*.	103
INPUT *variable*	[■] [INPUT] *variable* Recalls the *variable* to the X-register, displays the variable name along with the contents of the X-register, and halts program execution; pressing [R/S] (or [▼]) stores the number in the variable. (Used only in programs.)	77
IP	[■] [PARTS] {IP} *Integer part* of *x*.	67

Function Name	Keys and Description	Page
ISG *variable*	[■] [LOOP] {ISG} *variable* *Increment, Skip if Greater.* For control number *ccccccc.fffii* stored in *variable*, adds *ii* (incremental value) to *ccccccc* (counter value) and, if the result > *fff* (final value), skips the next program line.	101
LASTx	[■] [LASTx] Returns number stored in LAST X register.	41
LBL *letter*	[■] [LBL/RTN] {LBL} *label* Labels a program with a single letter for reference by XEQ, GTO, or FN operations. (Used only in programs.)	71
[■] [LBL/RTN]	Displays the menu for LBL, RTN, and PSE.	71
LN	[LN] *Natural logarithm.* Returns $\log_e x$.	55
LOG	[■] [LOG] *Common logarithm.* Returns $\log_{10} x$.	55
[■] [LOOP]	Displays the menu for DSE and ISG.	99
{L.R.}	[■] [STAT] {L.R.} Displays menu for linear regression.	158
m	[■] [STAT] {L.R.} {m} Returns the slope of the regression line: $[\Sigma(x_i - \bar{x})(y_i - \bar{y})] \div \Sigma(x_i - \bar{x})^2$.	159
[■] [MEM]	Displays the amount of available memory and the catalog menu.	33
[■] [MEM] {PGM}	Begins catalog of programs.	85
[■] [MEM] {VAR}	Begins catalog of variables.	49
[■] [MODES]	Displays the menu to set angular modes and the radix (. or ,).	29
n	[■] [STAT] {Σ} {n} Returns the number of sets of data points.	161
OCT	[■] [BASE] {OC} Selects Octal (base 8) mode.	144
[■] [OFF]	Turns the calculator off.	14
[■] [P↔RECT]	Displays the menu for converting between polar and rectangular coordinates.	60

Function Name	Keys and Description	Page
	[PARTS] Displays the menu for selecting parts of numbers.	67
Pn,r	[PROB] {Pn,r} *Permutations* of *n* items taken *r* at a time. Returns $n! \div (n - r)!$.	65
	[PRGM] Activates or cancels program entry (toggles).	72
	[PROB] Displays the menu for probability functions.	65
PSE	[LBL/RTN] {PSE} *Pause.* Halts program execution briefly to display *x*, then resumes. (Used only in programs.)	82
r	[STAT] {L.R.} {r} Returns the correlation coefficient between the *x*- and *y*-values: $\Sigma(x_i - \bar{x})(y_i - \bar{y}) \div \sqrt{\Sigma(x_i - \bar{x})^2 \times \Sigma(y_i - \bar{y})^2}$.	159
→RAD	[D↔RAD] {→RAD} *Degrees to radians.* Returns $(2\pi/360)x$.	64
{R}	[PROB] {R} Displays random-number menu.	65
RAD	[MODES] {RD} Selects Radians angular mode.	57
RADIX,	[MODES] { , } Selects the comma as the radix mark (decimal point).	29
RADIX.	[MODES] { . } Selects the period as the radix mark (decimal point).	29
RANDOM	[PROB] {R} {RANDOM} Returns a random number in the range $0 \leq x < 1$.	65
RCL *variable*	[RCL] *variable* *Recall.* Copies *variable* into the X-register.	48
RCL+ *variable*	[RCL] [+] *variable* Returns *x* + *variable*.	51
RCL− *variable.*	[RCL] [−] *variable.* Returns *x* − *variable*.	51

Function Name	Keys and Description	Page
RCL× *variable.*	[RCL] [×] *variable.* Returns *x* × *variable.*	51
RCL÷ *variable.*	[RCL] [÷] *variable.* Returns *x* ÷ *variable.*	51
RND	■ [PARTS] {RN} *Rounds x* to *n* decimal places (in FIX *n* display option) or to *n* + 1 significant digits (in SCI *n* or ENG *n* display option).	67
RTN	■ [LBL/RTN] {RTN} *Return.* Marks the end of a program; the program pointer returns to the top or to the calling routine.	71, 91
[R/S]	*Run/stop.* Begins program execution at the current program line *or* stops a running program.	82
R↓	[R↓] *Roll down.* Moves *t* to Z-register, *z* to Y-register, *y* to X-register, and *x* to T-register.	36
SCI *n*	■ [DISP] {SC}*n* Selects Scientific display with *n* decimal places, 0 ≤ *n* ≤ 11.	30
SEED	■ [PROB] {R} {SEED} Restarts the random-number sequence with the seed |x|.	65
SF *n*	■ [FLAGS] {SF}*n* *Sets flag n* (0 ≤ *n* ≤ 6), indicating "true."	98
■ [SHOW]	Shows the full mantissa (all 12 digits) of *x* (or the number in the current program line).	31
SIN	[SIN] *Sine.* Returns sin *x*.	57
SINH	■ [HYP] [SIN] *Hyperbolic sine.* Returns sinh *x*.	59
■ [SOLVE/∫]	Displays the menu for solving for an unknown and for integration.	111, 127

Function Name	Keys and Description	Page
SOLVE *variable*	■ [SOLVE/∫] {SOLVE} *variable* Solves the current function for the *variable*, using initial estimates in *variable* and *x*.	111
SQRT	[√x] *Square root* of *x*.	24
■ [STAT]	Displays the menu for statistical functions.	156
STO *variable*	[STO] *variable* *Store.* Copies *x* into *variable.*	48
STO+ *variable*	[STO] [+] *variable* Stores *variable* + *x* into *variable.*	50
STO− *variable*	[STO] [−] *variable* Stores *variable* − *x* into *variable.*	50
STO× *variable*	[STO] [×] *variable* Stores *variable* × *x* into *variable.*	50
STO÷ *variable*	[STO] [÷] *variable* Stores *variable* ÷ *x* into *variable.*	50
STOP	[R/S] Halts program execution and displays the X-register.	82
sx	■ [STAT] {Σ} {Σx} Returns the *standard deviation* of *x*-values: $\sqrt{\Sigma(x_i - \bar{x})^2 \div (n - 1)}$.	157
sy	■ [STAT] {Σ} {Σy} Returns the *standard deviation* of *y*-values: $\sqrt{\Sigma(y_i - \bar{y})^2 \div (n - 1)}$.	157
TAN	[TAN] *Tangent.* Returns tan *x*.	57
TANH	■ [HYP] [TAN] *Hyperbolic tangent.* Returns tanh *x*.	59
■ [TESTS]	Displays the menu of conditional tests.	96
VIEW *variable*	■ [VIEW] *variable* Displays the labeled contents of *variable* without recalling the value to the stack.	79

Function Name	Keys and Description	Page
XEQ *label*	[XEQ] *label* Executes the program identified by *label*.	75, 91
x^2	[■] [x^2] *Square* of *x*.	24
$\bar{x}$	[■] [STAT] {$\bar{x}$,$\bar{y}$} {$\bar{x}$} Returns the mean of *x* values: $\Sigma x_i \div n$.	156
$\hat{x}$	[■] [STAT] {L.R.} {$\hat{x}$} Given a *y*-value in the X-register, returns the *x-estimate* based on the regression line: $\hat{x} = (y - b) \div m$.	159
x!	[■] [PROB] {x!} *Factorial* (or gamma). Returns $(x)(x-1)\ldots(2)(1)$, or $\Gamma(x+1)$.	65
$\bar{x}$w	[■] [STAT] {$\bar{x}$,$\bar{y}$} {$\bar{x}$w} Returns the *weighted mean* of *x* values: $(\Sigma y_i x_i) \div \Sigma y_i$.	157
x<>y	[x≷y] *x exchange y*. Moves *x* to the Y-register and *y* to the X-register.	37
x<0?	[■] [TESTS] {x?0} {<0} If $x < 0$, executes the next program line; if $x \geq 0$, skips the next program line.	96
x<y?	[■] [TESTS] {x?y} {<y} If $x < y$, executes the next program line; if $x \geq y$, skips the next program line.	96
x=0?	[■] [TESTS] {x?0} {=0} If $x = 0$, executes the next program line; if $x \neq 0$, skips the next program line.	96
x=y?	[■] [TESTS] {x?y} {=y} If $x = y$, executes the next program line; if $x \neq y$, skips the next program line.	96
x>0?	[■] [TESTS] {x?0} {>0} If $x > 0$, executes the next program line; if $x \leq 0$, skips the next program line.	96

Function Name	Keys and Description	Page
x>y?	■ TESTS {x?y} {>y} If $x > y$, executes the next program line; if $x \leq y$, skips the next program line.	96
x≠0?	■ TESTS {x?0} {≠0} If $x \neq 0$, executes the next program line; if $x = 0$, skips the next program line.	96
x≠y?	■ TESTS {x?y} {≠y} If $x \neq y$, executes the next program line; if $x = y$, skips the next program line.	96
$\bar{y}$	■ STAT {x̄,ȳ} {ȳ} Returns the *mean* of y values: $\Sigma y_i \div n$.	156
$\hat{y}$	■ STAT {L.R.} {ŷ} Given an x-value in the X-register, returns the *y-estimate* based on the regression line: $\hat{y} = mx + b$.	159
y,x→θ,r	■ P↔RECT {y,x→θ,r} *Rectangular to polar.* Converts (x, y) to (r, θ).	61
y^x	y^x *Power.* Returns y raised to the x power.	56

Index

Page numbers in **bold** type indicate primary references. *To look up functions by name, use the function index that is before this index.*

Special Characters

A

B

C

D

E

F

G–I

K–L

M

N

O

P

Q–R

S

T

U–W

X–Z

Contacting Hewlett-Packard

For Information About Using the Calculator. If you have questions about how to use the calculator, first check the table of contents, the subject index, and "Answers to Common Questions" in appendix A. If you can't find an answer in the manual, you can contact the Calculator Support Department:

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

(503) 757-2004
8:00 a.m. to 3:00 p.m. Pacific time
Monday through Friday

For Service. If your calculator doesn't seem to work properly, see appendix A to determine if the calculator requires service. Appendix A also contains important information about obtaining service. If your calculator does require service, mail it to the Calculator Service Center:

Hewlett-Packard
Calculator Service Center
1030 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

(503) 757-2002

For Information About Hewlett-Packard Dealers, Products, and Prices. Call the following toll-free number:

(800) 752-0900

Contents

HEWLETT PACKARD

Reorder Number
00032-90039

00032-90065 English
Printed in U.S.A. 9/88