HEWLETT-PACKARD

HP-34C

OWNER'S HANDBOOK AND PROGRAMMING GUIDE

"The success and prosperity of our company will be assured only if we offer our customers superior products that fill real needs and provide lasting value, and that are supported by a wide variety of useful services, both before and after sale."

Statement of Corporate Objectives.
Hewlett-Packard

When Messrs. Hewlett and Packard founded our company in 1939, we offered one superior product, an audio oscillator. Today, we offer over 3500 quality products, designed and built for some of the world's most discerning customers.

Since we introduced our first scientific calculator in 1967, we've sold millions worldwide, both pocket and desktop models. Their owners include Nobel laureates, astronauts, mountain climbers, businesspersons, doctors, students, and homemakers.

Each of our calculators is precision crafted and designed to solve the problems its owner can expect to encounter throughout a working lifetime.

HP calculators fill real needs. And they provide lasting value.

HEWLETT hp PACKARD

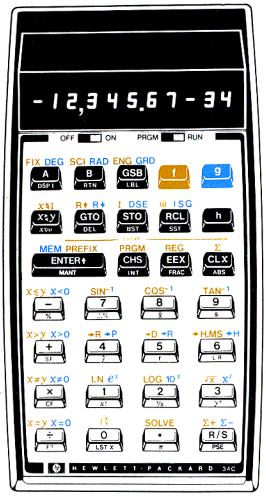
The HP-34C Programmable Scientific Calculator Owner's Handbook and Programming Guide

May 1979

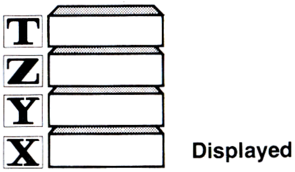
00034-90001

Contents

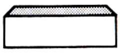
The HP-34C Programmable Scientific Calculator



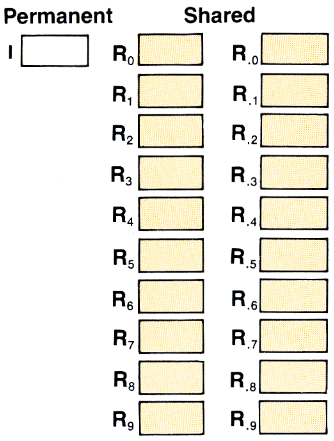
AUTOMATIC MEMORY STACK



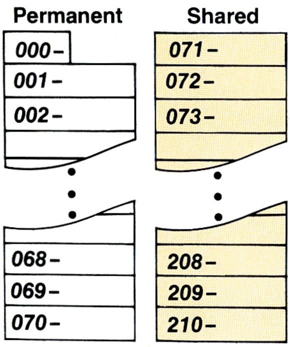
LAST X



STORAGE REGISTERS


Permanent
Shared
I
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9
R.0 R.1 R.2 R.3 R.4 R.5 R.6 R.7 R.8 R.9

PROGRAM MEMORY



The basic program memory-storage register allocation is 70 lines of programming and 72 data storage registers, plus the I-register. The calculator automatically converts one data storage register into seven lines of program memory, one register at a time, as you need them. Conversion begins with $R_{.9}$ and ends with R_0.

Function Key Index

OFF ON OFF-ON switch set to ON.

PRGM RUN PRGM-RUN switch set to RUN.

Function keys pressed from the keyboard execute individual functions as they are pressed. Input numbers and answers are displayed. Except where otherwise indicated, each function key listed below operates from the keyboard and as a recorded instruction in a program.

Prefix Keys

$\boxed{\text{f}}$ Pressed before a function key, selects gold function printed above that key.

$\boxed{\text{g}}$ Pressed before a function key, selects blue function printed above that key.

$\boxed{\text{h}}$ Pressed before a function key, selects black function printed on slanted face of that key.

CLEAR $\boxed{\text{PREFIX}}$ (nonprogrammable) cancels a partially entered instruction such as $\boxed{\text{f}}$, $\boxed{\text{f}}$ $\boxed{\text{SCI}}$, $\boxed{\text{g}}$, $\boxed{\text{h}}$, $\boxed{\text{h}}$ $\boxed{\text{SF}}$, $\boxed{\text{STO}}$ $\boxed{+}$, etc.

Digit Entry

$\boxed{\text{ENTER}\uparrow}$ Enters a copy of number in displayed X-register into Y-register. Used to separate numbers.

$\boxed{\text{CHS}}$ Changes sign of number or exponent of 10 in displayed X-register.

$\boxed{\text{EEX}}$ Enter exponent. After pressing, next digits keyed in are exponents of 10.

0 through 9 Digit keys.

$\boxed{\cdot}$ Decimal point.

Number Alteration

$\boxed{\text{INT}}$ Leaves only integer portion of number in displayed X-register by truncating fractional portion.

$\boxed{\text{FRAC}}$ Leaves only fractional portion of number in displayed X-register by truncating integer portion.

$\boxed{\text{ABS}}$ Gives absolute value of number in displayed X-register.

Stack Manipulation

$\boxed{R\downarrow}$ Rolls down contents of stack for viewing in displayed X-register.

$\boxed{R\uparrow}$ Rolls up contents of stack for viewing in displayed X-register.

$\boxed{x \gtrless y}$ Exchanges contents of X- and Y-stack registers.

$\boxed{\text{CLX}}$ Clears contents of displayed X-register to zero.

Storage

$\boxed{\text{STO}}$ Store. Followed by address key, stores displayed number in the storage register (R_0 through R_9, $R_{.0}$ through $R_{.9}$, I) specified. Also used to perform storage register arithmetic.

RCL Recall. Followed by address key, recalls number from storage register (R_0 through R_9, $R_{.0}$ through $R_{.9}$, I) specified into the displayed X-register.

LST x Recalls number displayed before the previous operation back into the displayed X-register.

CLEAR REG Clears contents of all storage registers (R_0 through R_9, $R_{.0}$ through $R_{.9}$, I) to zero.

Display Control

FIX Selects fixed point display.

SCI Selects scientific notation display.

ENG Selects engineering notation display.

DSP I Displays as many digits after the decimal point as are specified by the number in the I-register (0 through 9).

MANT Mantissa (nonprogrammable). Displays all 10 significant digits of the number in the X-register as long as the MANT key is pressed.

Percentage

Δ% Computes percent of change between number in Y-register and number in displayed X-register.

% Computes x% of value in the Y-register.

Mathematics

− + × ÷ Arithmetic operators.

$\sqrt{x}$ Computes square root of number in displayed X-register.

x^2 Computes square of number in displayed X-register.

x! Calculates factorial $x!$, or Gamma function $\Gamma(1+x)$.

1/x Computes reciprocal of number in displayed X-register.

π Places value of pi (3.141592654) into displayed X-register.

$\int_y^x$ Integrate. Computes definite integral $\int_y^x f(x)dx$ with expression $f(x)$ keyed into program memory.

SOLVE Solves for real root of equation $f(x)=0$, with expression $f(x)$ keyed into program memory.

Statistics

CLEAR Σ Clears statistical storage registers (R_0 through R_5).

Σ+ Accumulates numbers from X- and Y-registers into storage registers R_0 through R_5.

Σ− Subtracts x and y values from storage registers R_0 through R_5 for correcting Σ+ accumulations.

$\boxed{\bar{x}}$ Computes mean (average) of *x* and *y* values accumulated by $\boxed{\Sigma+}$.

$\boxed{s}$ Computes sample standard deviations of *x* and *y* values accumulated by $\boxed{\Sigma+}$.

$\boxed{\hat{y}}$ Linear estimate. Computes estimated value of *y* for a given value of *x* by method of least squares.

$\boxed{r}$ Correlation coefficient. Computes "goodness of fit" between the *x* and *y* values accumulated using $\boxed{\Sigma+}$ and the linear function that approximates them.

$\boxed{\text{L.R.}}$ Linear regression. Computes *y*-intercept and slope for linear function that best approximates *x* and *y* values accumulated using $\boxed{\Sigma+}$. The value of the *y*-intercept is placed in the X-register; the value of the slope is placed in the Y-register.

Polar/Rectangular Conversion

$\boxed{\to R}$ Converts polar magnitude *r* and angle *θ* in X- and Y-registers to rectangular *x* and *y* coordinates.

$\boxed{\to P}$ Converts *x*, *y* rectangular coordinates placed in X- and Y-registers to polar magnitude *r* and angle *θ*.

Trigonometry

$\boxed{\text{DEG}}$ Sets decimal degrees mode for trigonometric functions.

$\boxed{\text{RAD}}$ Sets radians mode for trigonometric functions.

$\boxed{\text{GRD}}$ Sets grads mode for trigonometric functions.

$\boxed{\text{SIN}}$ $\boxed{\text{COS}}$ $\boxed{\text{TAN}}$ Computes sine, cosine, or tangent of value in displayed X-register.

$\boxed{\text{SIN}^{-1}}$ $\boxed{\text{COS}^{-1}}$ $\boxed{\text{TAN}^{-1}}$ Computes arc sine, arc cosine, or arc tangent of number in displayed X-register.

$\boxed{\to R}$ Converts degrees to radians.

$\boxed{\to D}$ Converts radians to degrees.

$\boxed{\to \text{H.MS}}$ Converts decimal hours (or degrees) to hours, minutes, seconds (or degrees, minutes, seconds).

$\boxed{\to H}$ Converts hours, minutes, seconds (or degrees, minutes, seconds) to decimal hours (or degrees).

I-Register Control

$\boxed{x \rightleftarrows I}$ Exchanges contents of displayed X-register with those of the I-register.

$\boxed{x \rightleftarrows (i)}$ Exchanges contents of displayed X-register with those of the register addressed by the value stored in the I-register.

$\boxed{I}$ I-register. Storage register for increment/decrement operations and for indirect control of display and program execution.

[(i)] Indirect operations command. Used with [STO] and [RCL] for indirect data storage, recall, and storage register arithmetic.

[DSP I] Displays as many digits after the decimal point as are specified in the I-register.

[DSE] Decrement, skip when equal or less than. Subtracts specified decrement value from counter value. Skips one program line if new counter value is equal to or less than specified test value.

[ISG] Increment, skip when greater. Adds specified increment value to counter value. Skips one program line if new counter value is greater than specified test value.

Logarithmic and Exponential

[LN] Computes natural logarithm (base $e =$ 2.718281828...) of number in displayed X-register.

[e^x] Natural antilogarithm. Raises e (2.718281828) to power of number in displayed X-register.

[LOG] Computes common logarithm (base 10) of number in displayed X-register.

[10^x] Common antilogarithm. Raises 10 to power of number in displayed X-register.

[y^x] Raises number in Y-register to power of number in displayed X-register.

Program Control Index

Several of the following keys operate only in PRGM mode; others operate differently in PRGM mode than in RUN mode. For specific details of operation, consult the indicated pages.

MEM Displays current status of program memory/storage register allocation **(page 59).**

A B User-definable program keys for both program labels and execution **(page 68).**

0 1 2 3 4 5 6 7 8 9 Label designators. When preceded by LBL, define beginning of a routine **(page 68).**

LBL Label. When used with A, B, or 0 through 9, denote the beginning of a program or subroutine **(page 68).**

GTO Go to. Used with A, B, 0 through 9, or I. From the keyboard: causes calculator to search downward in program memory for designated label and halt. In a running program: causes calculator to transfer downward in program memory to designated label and resume program execution **(page 102).**

GSB Go to subroutine. Used with A, B, 0 through 9 or I. From the keyboard: causes calculator to search downward in program memory for designated label and begin program execution. In a running program: causes calculator to transfer downward in program memory to designated label and begin subroutine execution **(page 126).**

GTO • *nnn* Go to line number. Positions calculator to occupied line number specified by *nnn* **(page 85).**

BST Back step. Moves calculator back one line in occupied program memory **(page 84).**

SST Single-step. Moves calculator forward one line in occupied program memory **(page 84).**

DEL Delete. Used in PRGM mode to delete displayed instruction from program memory. All subsequent instructions are moved up one line **(page 85).**

CLEAR PRGM Clears all instructions in program memory and resets calculator to line 000 **(page 84).**

PSE Pause. Halts program execution for about one second to display contents of X-register, then resumes execution **(page 65).**

R/S Run/stop. Begins program execution from current line number in program memory. Stops execution if program is running **(page 62).**

RTN Return. Causes calculator to return from any line in occupied program memory to line 000, or from subroutine to appropriate line elsewhere in occupied program memory **(page 126).**

SF Set flag. Followed by flag designator (0, 1, 2, or 3) sets flag true **(page 120).**

CF Clear flag. Followed by flag designator (0, 1, 2, or 3) clears flag **(page 120).**

F? Is flag true? Followed by flag designator (0, 1, 2, or 3), tests designated flag. If flag is set (true) the calculator executes the instruction in the next line of program memory. If flag is cleared (false), calculator skips one line in program memory before resuming execution **(page 120).**

$x \leq y$ $x > y$ $x \neq y$ $x = y$

$x < 0$ $x > 0$ $x \neq 0$ $x = 0$

Conditionals. Each tests value in X-register against 0 or value in Y-register as indicated. If true, calculator executes instruction in next line of program memory. If false, calculator skips one line in program memory before resuming execution **(page 108).**

Section 1

Meet the HP-34C

Congratulations!

Your HP-34C Programmable Scientific Calculator with Continuous Memory is a truly unique and versatile calculating instrument. Using the Hewlett-Packard RPN logic system, your calculator can easily slice through the most difficult calculations with ease. It is without parallel:

As a scientific calculator. The HP-34C features a multiple-entry keyboard with each of the keys controlling up to four separate operations, ensuring maximum computing power.

As a problem-solving machine. Following step-by-step instructions in the *HP-34C Applications books,* you can key in programs from the areas of mathematics, engineering, statistics, surveying, and other fields and begin using your calculator. Immediately.

As a personal programmable calculator. The HP-34C is so easy to program and use that it requires no prior programming experience or knowledge of mysterious programming languages. Yet even computer experts can appreciate the calculator's programming features:

- Continuous Memory, allowing programs and data to be remembered by the calculator—even when the power switch is off.
- Automatic Memory Allocation: The basic 70 lines of program memory plus 21 storage registers reallocates in 7-line increments to as many as 210 lines of program memory plus 1 storage register—automatically—as needed.
- Fully merged prefix and function keys that mean more programming power per line.
- Easy-to-use editing features for correcting and modifying programs.
- Conditional and unconditional branching.

- Six levels of subroutines, 4 flags, 12 easily accessed and reusable labels.
- Direct or indirect storage, recall, branching, and subroutine calls.
- Powerful root-finding and numerical integration operations using the $\boxed{\text{SOLVE}}$ and $\boxed{\int_x^y}$ keys.

And in addition, the HP-34C can be operated from its rechargeable battery pack for *complete portability,* anywhere.

If you are new to HP calculators and their RPN logic system, you may want to carefully work through *Solving Problems With Your Hewlett-Packard Calculator* before consulting this handbook. Even if you already own another HP calculator, you may find some new features in the problem-solving book.

Now let's take a closer look at your calculator to see how easy it is to use, whether we solve a problem manually or use its programming power to solve the problem automatically.

Manual Problem Solving

Before continuing, you should be comfortable solving problems manually. If not, refer to the Getting Started section of *Solving Problems With Your Hewlett-Packard Calculator.*

Ready? Slide the OFF-ON switch to █▥ ON and be sure that the PRGM-RUN switch is in the █▥ RUN position. Now press $\boxed{\text{f}}$ $\boxed{\text{FIX}}$ 4 to be sure your HP-34C's display setting matches the setting used in the following pages.*

To see the close relationship between manual and programmed calculations, let's first calculate the solution to a problem. Then we'll use a program to calculate the solution to the same problem and others like it.

If you were to calculate the surface area of a sphere, you would use the formula $A = \pi d^2$ where:

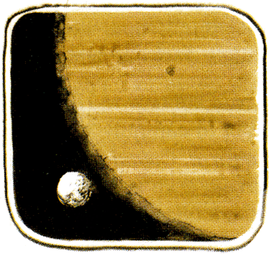
* The display setting used with examples in this handbook is always $\boxed{\text{FIX}}$ 4 unless otherwise indicated.

A is the surface area of the sphere.

d is the diameter of the sphere.

π is the value of pi, 3.141592654

Example: Ganymede, one of Jupiter's 12 moons, has a diameter of 3,200 miles. You can use the calculator to manually compute the surface area of Ganymede. Merely press the following keys in order. (Be sure the PRGM-RUN switch is in the RUN position.)



Keystrokes	Display	
3200	***3,200.***	Diameter of Ganymede.
$\boxed{g}$ $\boxed{x^2}$	***10,240,000.00***	Square of the diameter.
$\boxed{h}$ $\boxed{\pi}$	***3.1416***	The quantity π.
$\boxed{\times}$	***32,169,908.78***	Surface area of Ganymede in square miles.

Programmed Problem Solving

After calculating the surface area of Ganymede, suppose you decided you want to calculate the surface area of *each* moon. You could repeat the procedure you used for Ganymede 12 times, using a different diameter d each time. However, an easier and faster method is to create a *program* that will calculate the surface area of any sphere from its diameter rather than pressing all the keys for each moon.

To calculate the area of a sphere using a program, you first *write* the program, then you *key* the program into the calculator, and finally you *run* the program to calculate each answer.

Writing the Program. You have already written it! A program is little more than the series of keystrokes you would execute to calculate the solution manually. Two additional operations, a label and a return, are used to define the beginning and end of the program.

Loading the Program. To load the keystrokes of the program into the calculator:

1. Slide the PRGM-RUN switch to PRGM ▥█ (program).
2. Press $\boxed{f}$ CLEAR $\boxed{PRGM}$ to clear program memory.
3. Press the following keys in order. (When you are loading a program, the display gives you information that you will find useful later, but which you can ignore for now.)

Keystrokes	**Display**	
$\boxed{h}$ $\boxed{LBL}$ $\boxed{A}$	*001 – 25, 13, 11*	Defines the beginning of the program.
$\boxed{g}$ $\boxed{x^2}$	*002 – 15 3*	These are the same keys you
$\boxed{h}$ $\boxed{\pi}$	*003 – 25 73*	press to solve the problem
$\boxed{\times}$	*004 – 61*	manually.
$\boxed{h}$ $\boxed{RTN}$	*005 – 25 12*	Defines the end of the program.

The calculator will now remember this keystroke sequence.

Running the Program. To run the program to find the area of any sphere from its diameter:

1. Slide the PRGM-RUN switch to █▥ RUN .
2. Key in the value of the diameter.
3. Press $\boxed{A}$ to run the program.

When you press $\boxed{A}$, the sequence of keystrokes you loaded is automatically executed by the calculator, giving you the same answer you would have obtained manually.

For example, to calculate the surface area of Ganymede with a diameter of 3,200 miles:

Keystrokes	Display	
3200	***3,200.***	
[A]	***32,169,908.78***	Square miles.

With the program you have loaded, you can now calculate the surface area of any of Jupiter's moons—in fact, of *any* sphere—using its diameter. Leave the calculator in RUN mode and key in the diameter of each sphere for which you want the surface area, then press [A].

For example, compute the surface area of Jupiter's moon Io, with a diameter of 2,310 miles.

Keystrokes	Display	
2310 [A]	***16,763,852.56***	Square miles.

Now compute the surface area of the moons Europa, diameter 1,950 miles, and Callisto, diameter 3,220 miles.

Keystrokes	Display	
1950 [A]	***11,945,906.07***	Area of Europa in square miles.
3220 [A]	***32,573,289.27***	Area of Callisto in square miles.

Programming is *that* easy! The calculator remembers a series of keystrokes and then executes them whenever you wish. In fact, your HP-34C can remember up to 210 separate operations (and many more keystrokes, since many operations require two or three keystrokes).

What Continuous Memory Means to You

Your calculator contains Continuous Memory—one of the most advanced memory systems available in a scientific calculator. Continuous Memory means the program memory, all 21 storage registers, and the display mode stay "on" when your calculator is turned off. You can store your favorite program (or programs) for days or weeks!

Continuous Memory is especially convenient when you want to retain data, save battery life, or customize your calculator (e.g., if you use 20% of your programs in 80% of your calculations.) You save considerable time because you don't have to key in those common programs again and again—they are stored in your calculator. Continuous Memory reduces human entry errors too; fewer keystrokes mean fewer chances of making inadvertent errors. Perhaps the most important advantage of Continuous Memory is that it enables you to customize or personalize your calculator. The easiest way to customize your calculator is to make a list of the problems you encounter most frequently, rate them in order of priority, then write and save the specialized programs for those problems. Whenever you encounter a repetitive problem set, you just write the program once, then use it at different times. You can even preserve one or two favorite programs in the calculator.

Besides saving programs, Continuous Memory lets you save data in up to 21 storage registers, depending on current program memory/data storage allocation. Constants, accumulations, and intermediate answers can be retrieved whenever you need them. And because display mode is also saved in Continuous Memory, your HP-34C "wakes up" in whatever [FIX], [SCI], or [ENG] setting you last used.

Continuous Memory helps save battery life because you don't have to keep the calculator turned on to save programs or data between calculations. And if your calculator is left off, Continuous Memory can store your programs and data for 1 month or longer. When you do use your calculator, keying in fewer programs means less time that the display is on—hence, less battery drain.

Specific Features of the HP-34C

Most of the features found on the HP-34C are discussed in *Solving Problems With Your Hewlett-Packard Calculator*. However, several features unique to the HP-34C (or new to HP calculators) are discussed in the following pages.

Keyboard Operation

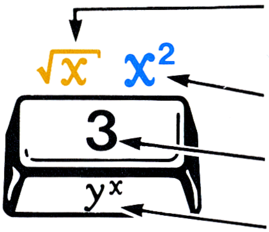
Most keys on the HP-34C keyboard perform three or four functions. One function is indicated by the symbol on the horizontal key face, while another is printed in black on the slanted key face. A third function is indicated by the gold symbol printed above the key. On keys designed with four functions, the last function is indicated by the appropriate blue symbol, also printed above the key.

To select the function on the horizontal face of a key, press the key.

To select the function printed in black on the slanted face of a key, press the black prefix key [h], then press the function key.

To select the function printed in gold above a key, press the gold prefix key [f], then press the function key.

To select the function printed in blue above a key, press the blue prefix key [g], then press the function key.



To execute this function, press [f] [$\sqrt{x}$].

To execute this function, press [g] [x^2].

To execute this function, simply press 3.

To execute this function, press [h] [y^x].

Notice that for all four-function keys *except* $\overset{\text{MEM PREFIX}}{\boxed{\text{ENTER}\uparrow}}$, the function printed in gold is above and to the left of the key and the function printed in blue is above and to the right of the key.

Storage Registers and Program Memory

In addition to the four-register stack and the LAST X register, your HP-34C features a shared program and data storage memory that is controlled automatically. The HP-34C's basic program memory/data storage allocation is 70 lines of program memory and 20 data storage registers, plus the I-register. According to your programming needs, one or more of the data storage registers can be automatically exchanged for seven lines each of program memory. And, by pressing $\boxed{\text{g}}$ $\boxed{\text{MEM}}$ your HP-34C will even tell you the current program memory storage register allocation at any time! We will cover this important subject in detail when we discuss programming.

Storing and Recalling Numbers. Your HP-34C's twenty data storage registers are denoted R_0 through R_9 and $R_{.0}$ through $R_{.9}$ (plus the I-register, which we'll cover later). As you learned in *Solving Problems With Your Hewlett-Packard Calculator,* a copy of the number in the displayed X-register can be stored in registers R_0 through R_9 by pressing $\boxed{\text{STO}}$ *(store)* and the number key of the register address (0-9). A copy of a number in any register R_0 through R_9 can be recalled to the displayed X-register by pressing $\boxed{\text{RCL}}$ *(recall)* and the number key of the register address (0-9). Notice, however, that store or recall operations involving registers $R_{.0}$ through $R_{.9}$ use an additional keystroke, $\boxed{\cdot}$, to correspond to the decimal point in these register names. For example, to store a copy of π in register $R_{.5}$:

Keystrokes	Display	
$\boxed{\text{h}}$ $\boxed{\pi}$	3.1416	
$\boxed{\text{STO}}$ $\boxed{\cdot}$ 5	3.1416	A copy of π is now stored in $R_{.5}$.

To recall a copy of π from $R_{.5}$:

Keystrokes	Display
CLX	**0.0000**
RCL • 5	**3.1416**

Storage Register Arithmetic. Registers R_0 through R_9 in your HP-34C are used for the direct storage register arithmetic operations described in *Solving Problems With Your Hewlett-Packard Calculator*. However, all storage registers (R_0 through R_9, $R_{.0}$ through $R_{.9}$, and I) can be used to perform indirect storage register arithmetic (we'll cover this subject later, in section 7).

Number Alteration Keys

Besides CHS, your HP-34C has three keys provided for altering numbers: ABS , INT , and FRAC .

Absolute Value

Some calculations require the absolute value, or magnitude, of a number. To obtain the absolute value of the number in the displayed X-register, press h followed by the ABS *(absolute value)* key. For example, to find the absolute value of –3.

Keystrokes	Display
3 CHS	**-3.**
h ABS	**3.0000**

Integer Portion of a Number

To extract and display the integer portion of a number, press the h prefix key followed by the INT *(integer)* key. For example, to display only the integer portion of the number 111.222.

Keystrokes	Display	
111.222	**111.222**	
[h] [INT]	**111.0000**	Only the integer portion remains.
333.444 [CHS]	**-333.444**	
[h] [INT]	**-333.0000**	Again, only the integer portion remains.

When [h] [INT] is pressed, the fractional portion of the number is replaced by zero. The sign is unaffected. The original number, of course, is preserved in the LAST X register.

[h] [LST x]	**-333.4440**	The original number.

Fractional Portion of a Number

To extract and display only the fractional portion of a number, press the [h] prefix key followed by the [FRAC] *(fraction)* key. For example, to see the fractional portion of 555.666.

555.666	**555.666**	
[h] [FRAC]	**0.6660**	Only the fractional portion of the number remains.
777.888 [CHS]	**-777.888**	
[h] [FRAC]	**-0.8880**	Again, only the fractional portion remains.

When the [h] [FRAC] is pressed, the integer portion of the number is replaced by zero. The sign is unaffected. The original number is preserved in the LAST X register.

[h] [LST x]	**-777.8880**

Mathematical Functions

Factorial

When the number in the X-register is a nonnegative integer n, pressing $\boxed{x!}$ gives you the factorial of n, which is denoted $n!$ and defined as the product of the integers from 1 to n. This function enables you to quickly and easily solve permutations and combinations.

Example: Willie's Widget Works wants a photograph of its product line for advertising. How many different ways can the photographer arrange their eight widget models?

Solution: The number of arrangements is given by

$$8! = 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$$

Keystrokes	Display	
8 $\boxed{h}$ $\boxed{x!}$	***40,320.0000***	The photographer can arrange the widgets 40,320 different ways.

Example: The photographer looks through her viewfinder and decides that she can show only five widgets if her camera is to capture the intricate details of the widgets on film. How many different sets of five widgets can she select from the eight?

Solution: The number of sets is given by

$$\frac{8!}{(8-5)!\,5!}$$

Keystrokes	Display	
8 $\boxed{h}$ $\boxed{x!}$	**40,320.0000**	
8 $\boxed{ENTER\uparrow}$	**8.0000**	
5 $\boxed{-}$	**3.0000**	
$\boxed{h}$ $\boxed{x!}$	**6.0000**	
5 $\boxed{h}$ $\boxed{x!}$	**120.0000**	
$\boxed{\times}$	**720.0000**	
$\boxed{\div}$	**56.0000**	The photographer can select 56 different sets of five widgets.

Gamma Function

The $\boxed{x!}$ key can also be used* to calculate the Gamma function, denoted by $\Gamma(x)$, which occurs in certain problems in advanced mathematics and statistics. Pressing $\boxed{x!}$ gives you $\Gamma(x + 1)$. To calculate the Gamma function of any number, therefore, subtract 1 from the number, then with the result in the X-register, press $\boxed{h}$ $\boxed{x!}$.

Example: Calculate $\Gamma(2.7)$.

Keystrokes	Display	
2.7	**2.7**	Key in number.
$\boxed{ENTER\uparrow}$ 1 $\boxed{-}$	**1.7000**	Subtract 1.
$\boxed{h}$ $\boxed{x!}$	**1.5447**	$\Gamma(2.7)$.

Example: Calculate $\Gamma(-2.7)$.

Keystrokes	Display	
2.7 $\boxed{CHS}$	**-2.7**	Key in number.
$\boxed{ENTER\uparrow}$ 1 $\boxed{-}$	**-3.7000**	Subtract 1.
$\boxed{h}$ $\boxed{x!}$	**-0.9311**	$\Gamma(-2.7)$.

* The $\boxed{x!}$ key can be used for both the factorial and Gamma functions because when x is a nonnegative integer n, $\Gamma(x + 1) = \Gamma(n + 1) = n!$. The Gamma function can be regarded as a generalization of the factorial function, since the number in the X-register is not limited to nonnegative integers. Conversely, the factorial function can be regarded as a special case of the Gamma function.

Since $\Gamma(x)$ is not defined when x is a negative integer or 0, $\Gamma(x + 1)$, the value returned by [x!], is not defined when x is a negative integer. As x approaches these values, the magnitude of $\Gamma(x + 1)$ increases without limit. Since the largest number your HP-34C can calculate is $9.999999999 \times 10^{99}$, if you press [h] [x!] with a negative integer in the X-register the calculator will display the overflow indication **-9.999999 99.** Although $\Gamma(x + 1)$ as x approaches a negative integer may be positive or negative, depending on the value of x, the calculator always displays a minus sign in the overflow display when x is a negative integer. This differentiates the value from the overflow display **9.999999 99** for very large positive values of x, at which $\Gamma(x)$ increases without limit but is always positive.

Percent Difference

The [Δ%] operation gives you the *percent difference*—that is, the relative increase or decrease—between two numbers. To find the percent difference:

1. Key in the base number (typically, the number that occurs first in time).
2. Press [ENTER↑].
3. Key in the second number.
4. Press [h] [Δ%].

The formula used is: $\Delta\% = \dfrac{100(x - y)}{y}$.

Using the above order of entry, a positive result signifies an increase, while a negative result signifies a decrease.

Example: Silas Silversaver's coin collection was appraised in 1974 at $475. An appraisal in 1979 valued the collection at $735. By what percent did the value of the collection increase from 1974 to 1979?

Keystrokes	Display	
475 ENTER↑	*475.0000*	
735 h Δ%	*54.7368*	Percent increase.

Statistical Functions

Accumulations

Pressing the Σ+ key computes certain important sums and products of the values in the X- and Y-registers. The results are automatically accumulated in storage registers R_0 through R_5. Before you start to calculate accumulations with a new set of x and y values, you should first clear these registers by pressing f CLEAR Σ. Then, do the following for each pair of x and y values in your data:

1. Key the y value into the X-register.
2. Press ENTER↑ to raise the y value into the Y-register.
3. Key the x value into the X-register.
4. Press f Σ+.

If your statistics problem involves only one variable (x) instead of two (x and y), the procedure is similar. First clear statistical storage registers R_0 through R_5. In addition, if the contents of the Y-register are not zero, you should clear the Y-register also. (A nonzero number in the Y-register during *one*-variable calculations of s, r, $L.R.$, or $\hat{y}$ may result in a display of ***Error 3***.) Pressing f CLEAR REG will clear registers R_0 through R_5, but will also clear registers R_6 through R_9, $R_{.0}$ through $R_{.9}$, and I. Therefore, if there are numbers stored in these other registers that you want to save, you should press the keys f CLEAR Σ instead of f CLEAR REG. After clearing the registers, do the following for each value of x in your data:

1. Key the number into the X-register.
2. Press f Σ+ .

Each time you press f Σ+ , the following operations are performed:

1. The number in the X-register is added to the contents of storage register R_1.
2. The square of the number in the X-register is added to the contents of storage register R_2.
3. The number in the Y-register is added to the contents of storage register R_3.
4. The square of the number in the Y-register is added to the contents of storage register R_4.
5. The number in the Y-register is multiplied by the number in the X-register, and the product is added to the contents of storage register R_5.
6. The number 1 is added to the contents of storage register R_0. The result—the number of (x,y) data pairs accumulated so far—is copied into the displayed X-register.

After you press f Σ+ , the number previously in the X-register is placed in the LAST X register. The number previously in the Y-register is not changed.

To summarize, this is where the statistical accumulations are stored inside your calculator:

Register	Contents
R_0	n: number of data pairs accumulated.
R_1	Σx: summation of x values.
R_2	Σx^2: summation of squares of x values.
R_3	Σy: summation of y values.
R_4	Σy^2: summation of squares of y values.
R_5	Σxy: summation of products of x values and y values.

Some sets of data consist of x or y values that all differ from some number by a comparatively small amount. You can maximize the precision of any statistical calculation involving such data by entering into the calculator only the differences between each value and a number approximating the average of the values. When you do this, this number must be added to the result of calculating $\bar{x}$, $\hat{y}$, or the y-intercept of *L.R.* For example, if your x values consist of 665999, 666000, and 666001, you should enter the data as −1, 0, and 1. If afterwards you calculate $\bar{x}$, add 666000 to the answer. In some cases the calculator cannot compute s, r, *L.R.*, or $\hat{y}$ with data values that are too close to each other, and if you attempt to do so the calculator will display ***Error 3***. This will not happen, however, if you normalize the data as described above.

Note: Unlike storage register arithmetic, the $\boxed{\Sigma+}$ and $\boxed{\Sigma-}$ operations allow overflow to occur in storage registers R_0 through R_5 without indicating ***Error 1*** in the display. (i.e., when executing $\boxed{\Sigma+}$or $\boxed{\Sigma-}$would result in an overflow in any statistics register, $9.999999999 \times 10^{99}$ is placed in that register without interrupting normal operation.)

To use any of the accumulations, you can recall the contents of the desired storage register into the displayed X-register by pressing $\boxed{\text{RCL}}$ followed by the number of the register. If this is done immediately after pressing $\boxed{\text{f}}$ $\boxed{\Sigma+}$ (or $\boxed{\text{g}}$ $\boxed{\Sigma-}$), the accumulation recalled is written over the number of data pair entries (n) in the display.

If you want to use both Σx and Σy, press RCL f Σ+. This simultaneously copies Σx from R_1 into the displayed X-register and copies Σy from R_3 into the Y-register. If this is done immediately after pressing f Σ+, g Σ−, CLx, or ENTER↑, the number in the Y-register is first lifted into the Z-register. Otherwise, the numbers in the X- and Y-registers are first lifted into the Z- and T-registers, respectively.

Example: Find Σx, Σx^2, Σy, Σy^2, and Σxy for the paired values of x and y listed below.

y	7	5	9
x	5	3	8

Keystrokes	**Display**	
f CLEAR Σ	0.0000	Clear statistical storage registers. (Display shown assumes no results remain from previous calculations.)
7 ENTER↑	7.0000	
5 f Σ+	1.0000	First pair is accumulated; $n = 1$.
5 ENTER↑	5.0000	
3 f Σ+	2.0000	Second pair is accumulated; $n = 2$.
9 ENTER↑	9.0000	
8 f Σ+	3.0000	Third pair is accumulated; $n = 3$.
RCL 1	16.0000	Sum of x values from register R_1.
RCL 2	98.0000	Sum of squares of x values from register R_2.
RCL 3	21.0000	Sum of y values from register R_3.
RCL 4	155.0000	Sum of squares of y values from register R_4.
RCL 5	122.0000	Sum of products of x and y values from register R_5.
RCL 0	3.0000	Number of entries ($n = 3$) from register R_0.

Deleting and Correcting Data

If you key in an incorrect value and have not yet pressed [f] [Σ+], press [CLx] and key in the correct value.

If you want to change one of the values, or if you discover after pressing [f] [Σ+] that one of the values was erroneous, you can correct the accumulations by using the [Σ-] *(summation minus)* key as follows:

1. Key the *incorrect* data pair into the X- and Y-registers. (You can use [LST x] to return a single incorrect data value to the displayed X-register.)
2. Press [g] [Σ-] to delete the incorrect data.
3. Key in the correct values for x and y. If one value of an (x,y) data pair is incorrect, you must delete and reenter both values.
4. Press [f] [Σ+].

For example, if the last data pair (8,9) in the previous example should have been (8,6), you could correct the accumulation as follows:

Keystrokes	**Display**	
9 [ENTER↑]	**9.0000**	Incorrect y value is entered again.
8	**8.**	Correct x value is entered again.
[g] [Σ-]	**2.0000**	Numbered of entries (n) is now two.
6 [ENTER↑]	**6.0000**	Correct y value is entered.
8	**8.**	X value is entered again.
[f] [Σ+]	**3.0000**	Number of entries is again three.

Note: Although $\boxed{g}$ $\boxed{\Sigma-}$ can be used to delete an erroneous (*x*, *y*) pair, it will not delete any rounding errors that may have occured when that pair was added into accumulating registers R_1 through R_5. Consequently, subsequent results may be different than they would have been if the erroneous pair (*x*,*y*) had not been entered via $\boxed{f}$ $\boxed{\Sigma+}$ and then deleted via $\boxed{g}$ $\boxed{\Sigma-}$. However, the difference will not be serious unless the erroneous pair (*x*,*y*) have a magnitude that is enormous compared with the correct pair; and in such a case it may be wise to start over again and re-enter the data again (and more carefully!).

Mean

Pressing $\boxed{\bar{x}}$ computes the arithmetic *mean* (average) of *x* and *y* values accumulated in registers R_1 and R_3, respectively.

When you press $\boxed{h}$ $\boxed{\bar{x}}$

1. The contents of the stack registers are lifted just as they are when you press $\boxed{RCL}$ $\boxed{f}$ $\boxed{\Sigma+}$, as described on page 30.

2. The mean of the *x* values ($\bar{x}$) is calculated using the data accumulated in registers R_1 (Σx) and R_0 (n) according to the formula:

$$\bar{x} = \frac{\Sigma x}{n}$$

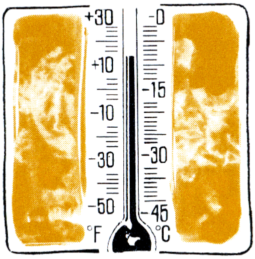
The resultant value for $\bar{x}$ appears in the displayed X-register.

3. The mean of the *y* values ($\bar{y}$) is calculated using the data accumulated in registers R_3 (Σy) and R_0 (n) according to the formula:

$$\bar{y} = \frac{\Sigma y}{n}$$

The resultant value for $\bar{y}$ is available in the Y-register of the stack.

Example: Below is a chart of daily high and low temperatures for a winter week in Fairbanks, Alaska. What are the average high and low temperatures for the week selected?



	Sun	Mon	Tues	Wed	Thurs	Fri	Sat
High	6	11	14	12	5	−2	−9
Low	−22	−17	−15	−9	−24	−29	−35

Keystrokes	**Display**	
$\boxed{f}$ CLEAR $\boxed{\Sigma}$	***0.0000***	Accumulation registers cleared. (Display shown assumes no results remain from previous calculations.)
6 $\boxed{\text{ENTER}\uparrow}$ 22	***22.***	
$\boxed{\text{CHS}}$ $\boxed{f}$ $\boxed{\Sigma+}$	***1.0000***	Number of data pairs (*n*) is now 1.
11 $\boxed{\text{ENTER}\uparrow}$ 17	***17.***	
$\boxed{\text{CHS}}$ $\boxed{f}$ $\boxed{\Sigma+}$	***2.0000***	Number of data pairs (*n*) is now 2.
14 $\boxed{\text{ENTER}\uparrow}$ 15	***15.***	
$\boxed{\text{CHS}}$ $\boxed{f}$ $\boxed{\Sigma+}$	***3.0000***	
12 $\boxed{\text{ENTER}\uparrow}$ 9	***9.***	
$\boxed{\text{CHS}}$ $\boxed{f}$ $\boxed{\Sigma+}$	***4.0000***	
5 $\boxed{\text{ENTER}\uparrow}$ 24	***24.***	
$\boxed{\text{CHS}}$ $\boxed{f}$ $\boxed{\Sigma+}$	***5.0000***	
2 $\boxed{\text{CHS}}$ $\boxed{\text{ENTER}\uparrow}$	***−2.0000***	
29 $\boxed{\text{CHS}}$ $\boxed{f}$ $\boxed{\Sigma+}$	***6.0000***	

Keystrokes	Display	
9 CHS ENTER↑	-9.0000	
35 CHS f Σ+	7.0000	Number of data pairs (n) is now 7.
h x̄	-21.5714	Average low temperature.
x⇄y	5.2857	Average high temperature.

Standard Deviation

Pressing h s computes the *standard deviation* (a measure of dispersion around the mean) of the accumulated data. The formulas used by the HP-34C to compute s_x, the standard deviation of the accumulated x values, and s_y, the standard deviation of the accumulated y values, are:

$$s_x = \sqrt{\frac{n\Sigma x^2 - (\Sigma x)^2}{n(n-1)}} \qquad s_y = \sqrt{\frac{n\Sigma y^2 - (\Sigma y)^2}{n(n-1)}}$$

These formulas give the *best estimates* of the *population* standard deviations from the *sample* data. Consequently, the standard deviation given by these formulas is termed by convention the *sample* standard deviation.

When you press h s:

1. The contents of the stack registers are lifted just as they are when you press RCL f Σ+, as described on page 30.
2. The standard deviation of the x values (s_x) is calculated using the data accumulated in registers R_2 (Σx^2), R_1 (Σx), and R_0 (n) according to the formula shown above. The resultant value for s_x appears in the displayed X-register.
3. The standard deviation of the y values (s_y) is calculated using the data accumulated in registers R_4 (Σy^2), R_3 (Σy), and R_0 (n) according to the formula shown above. The resultant value for s_y is available in the Y-register.

Example: Norman Numbercruncher, a rising young math professor at Mammoth University, has developed a new test for measuring the mathematical abilities of college freshmen. To evaluate its effectiveness, he administers the test to the 746 students in Calculus I. Exhausted after grading the tests, Numbercruncher decides to randomly select 8 of the 746 tests and estimate the standard deviation of all the scores from the sample of 8. The scores on the tests selected were 79, 94, 68, 86, 82, 78, 83, and 89. What standard deviation does Numbercruncher calculate?

Keystrokes	**Display**	
CLx ENTER↑	*0.0000*	Clear displayed X-register and Y-register.
f CLEAR Σ	*0.0000*	Clear statistical registers
79 f Σ+	*1.0000*	First score is entered. Notice that since this problem involves only one variable, you don't have to enter a *y*-value into the Y-register using the ENTER↑ key.
94 f Σ+	*2.0000*	Display shows number of scores entered so far.
68 f Σ+	*3.0000*	
86 f Σ+	*4.0000*	
82 f Σ+	*5.0000*	
78 f Σ+	*6.0000*	
83 f Σ+	*7.0000*	
89 f Σ+	*8.0000*	Last score in sample.
h s	*7.8365*	Standard deviation estimated for the 746 students based on sample of 8.

When your data constitutes not just a sample of a population but rather *all* of the population, the standard deviation of the data is the *true* population standard deviation (denoted σ). The formula for the true population standard deviation differs by a factor of $[(n-1)/n]^{1/2}$ from the formula used for the s function. The difference between the values is small, and for most applications can be ignored. Nevertheless, if you want to calculate the exact value of the population standard deviation for an entire population, you can easily do so with just a few keystrokes on your HP-34C. Simply add, using the f Σ+ key, the mean (x) of the data to the data and then press h s. The result will be the true population standard deviation of the original data.

Example: Suppose the data from the previous example represented all the final exam scores from Numbercruncher's seminar on transcendental functions. Since this is the first time Numbercruncher has given this seminar, he wants to calculate the standard deviation of the test scores to determine how good his exam was. Numbercruncher takes his calculator in hand, enters the data, then proceeds as follows:

Keystrokes	**Display**	
h $\bar{x}$	*82.3750*	Mean of scores.
f Σ+	*9.0000*	Mean is added to data. Display shows nine total entries.
h s	*7.3304*	Standard deviation for all scores on final exam.

Linear Regression

Linear regression is a statistical method for finding a straight line that best fits a set of data points, thus providing a relationship between two variables. After a group of data points has been totaled in registers R_0 through R_5, you can calculate the coefficients of the linear equation $y = Ax + B$ using the least squares method by pressing h L.R.. (Naturally, at least two data points must be in the calculator before a least squares line can be fitted to them.)

To use the linear regression function on your HP-34C, first key in a series of data points using the f Σ+ key. Then press h L.R..

When you press [h] [L.R.]:

1. The contents of the stack registers are lifted just as they are when you press [RCL] [f] [Σ+], as described on page 30.

2. The slope (A) of the least squares line of the data is calculated using the equation:

$$A = \frac{n\ \Sigma xy - \Sigma x\ \Sigma y}{n\ \Sigma x^2 - (\Sigma x)^2}$$

The slope is available in the Y-register of the stack.

3. The y-intercept (B) of the least squares line of the data is calculated using the equation:

$$B = \frac{\Sigma y\ \Sigma x^2 - \Sigma x\ \Sigma xy}{n\ \Sigma x^2 - (\Sigma x)^2}$$

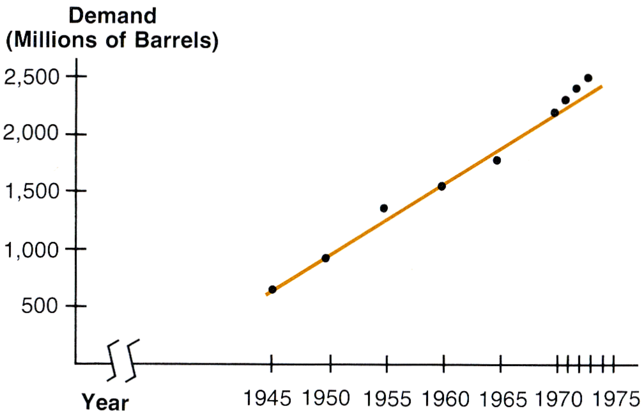
The y-intercept appears in the displayed X-register of the stack.

To use the value for A or to bring it into the displayed X-register, simply exchange the stack contents with the [x⇄y] key.

Example: Big George Gusher, owner-operator of the Gusher Oil Company, wishes to know the slope and y-intercept of a least squares line for the consumption of motor fuel in the United States against time since 1945. He knows the data given in the following table.

Motor Fuel Demand (Millions of Barrels)	696	994	1,330	1,512	1,750	2,162	2,243	2,382	2,484
Year	1945	1950	1955	1960	1965	1970	1971	1972	1973

Solution: Gusher *could* draw a plot of motor fuel demand against time like the one shown below.



However, with his HP-34C, Gusher has only to key the data into the calculator using the key, then press $\boxed{h}$ $\boxed{L.R.}$.

Keystrokes	Display	
$\boxed{f}$ CLEAR $\boxed{\Sigma}$	*0.0000*	Clear statistical storage registers. (Display shown assumes no results remain from previous calculations.)
696 $\boxed{ENTER\uparrow}$	*696.0000*	
1945 $\boxed{f}$ $\boxed{\Sigma+}$	*1.0000*	
994 $\boxed{ENTER\uparrow}$	*994.0000*	
1950 $\boxed{f}$ $\boxed{\Sigma+}$	*2.0000*	
1330 $\boxed{ENTER\uparrow}$	*1,330.0000*	
1955 $\boxed{f}$ $\boxed{\Sigma+}$	*3.0000*	
1512 $\boxed{ENTER\uparrow}$	*1,512.0000*	
1960 $\boxed{f}$ $\boxed{\Sigma+}$	*4.0000*	

1750 ENTER↑	**1,750.0000**	
1965 f Σ+	**5.0000**	
2162 ENTER↑	**2,162.0000**	
1970 f Σ+	**6.0000**	
2243 ENTER↑	**2,243.0000**	
1971 f Σ+	**7.0000**	
2382 ENTER↑	**2,382.0000**	
1972 f Σ+	**8.0000**	
2484 ENTER↑	**2,484.0000**	
1973 f Σ+	**9.0000**	All data pairs have been keyed in.
h L.R.	**–118,290.6295**	The y-intercept of the line.
x⇄y	**61.1612**	Slope of the line.

Linear Estimation

With data accumulated in registers R_0 through R_5, a predicted value for y (denoted $\hat{y}$) can be calculated by keying in a new value for x and pressing h $\hat{y}$.

For example, with data intact from the previous example in registers R_0 through R_5, if Gusher wishes to predict the demand for motor fuel for the years 1980 and 2000, he keys in the new x value and presses h $\hat{y}$.

Keystrokes	**Display**	
1980 h $\hat{y}$	**2,808.6264**	Predicted demand in millions of barrels for the year 1980.
2000 h $\hat{y}$	**4,031.8512**	Predicted demand in millions of barrels for the year 2000.

Correlation Coefficient

Both linear regression and linear estimation presume that the relationship between the x and y data values can be approximated, to some degree, by a linear function (i.e., a straight line). You can use [r] *(correlation coefficient)* to determine how closely your data "fits" a straight line. The correlation coefficient can range from $r = +1$ to $r = -1$. At $r = +1$, the data falls exactly onto a straight line with positive slope, while at $r = -1$, the data falls exactly onto a straight line with negative slope. At $r = 0$, the data cannot be approximated at all by a straight line.

For example, to calculate the correlation coefficient for the example above:

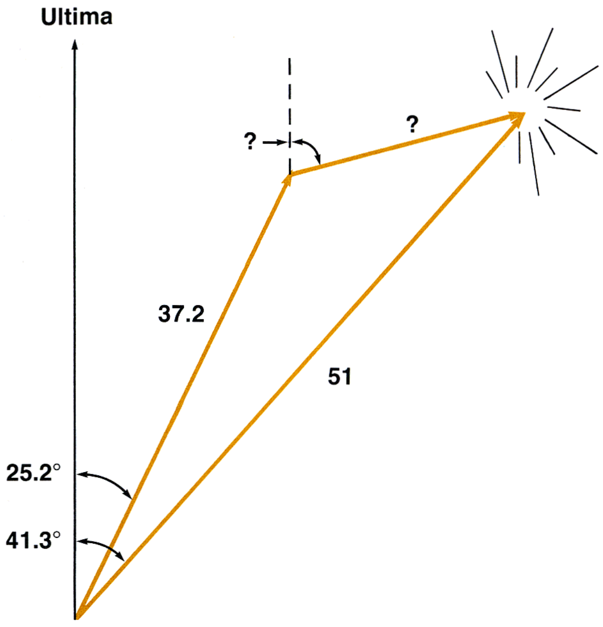
Keystrokes	Display	
[h] [r]	**0.9931**	The data approximates a straight line very closely.

Vector Arithmetic

You can add or subtract vectors with your HP-34C by using [Σ+] and [Σ−] in conjunction with [→R] and [→P].

Example: Federation starship *Felicity* has emerged victorious from a furious battle with the starship *Thanatos* from the renegade planet Maldek. However, its automatic pilot is kaput, and its main thrust engine is locked on at 37.2 meganewtons directed along a angle of 25.2° from the star Ultima. Consulting the ship's star map, the navigator reports a hyperspace entrance vector of 51 meganewtons at an angle of 41.3° from Ultima. To what thrust and angle should the auxiliary engine be set, for *Felicity* to achieve alignment with the hyperspace entrance vector?

Solution: The required thrust vector of the auxiliary engine is equal to the hyperspace entrance vector minus the thrust vector of the main engine. The vectors are converted to rectangular coordinates using [f] [→R], and their difference is calculated using [f] [Σ+] and [g] [Σ−]. This difference is recalled to the X- and Y-registers using [RCL] [f] [Σ+]. Then, these rectangular coordinates of the auxiliary engine thrust vector are converted to polar coordinates using [g] [→P].



Keystrokes	Display	
f CLEAR Σ	*0.0000*	Clear statistical registers. (Display shown assumes no results remain from previous calculations.)
g DEG	*0.0000*	Ensures that trigonometric mode is set to degrees.
41.3 ENTER↑	*41.3000*	Enter angle of hyperspace entrance vector into Y-register.
51 f →R	*38.3145*	Enter magnitude of hyperspace entrance vector into X-register and convert to rectangular coordinates.

Keystrokes	Display	
f Σ+	**1.0000**	Rectangular coordinates of hyperspace entrance vector accumulated in registers R_1 and R_3.
25.2 ENTER↑	**25.2000**	Enter angle of main engine thrust vector into Y-register.
37.2 f →R	**33.6596**	Enter magnitude of main engine thrust vector into X-register and convert to rectangular coordinates.
g Σ−	**0.0000**	Subtract rectangular coordinates of main engine thrust vector from rectangular coordinates of hyperspace entrance vector in registers R_1 and R_3 into X-register and Y-register.
RCL f Σ+	**4.6549**	Recall rectangular coordinates of auxiliary engine thrust vector from registers R_1 and R_3 into X-register and Y-register.
g →P	**18.4190**	Convert to polar coordinates. Display shows required magnitude, in meganewtons, of auxiliary engine thrust vector.
x⇄y	**75.3613**	Required angle of auxiliary engine thrust vector.

Section 3

Simple Programming

What Is a Program?

A program is a sequence of keystrokes that is remembered by the calculator. You can execute a given program as often as you like—typically with just one keystroke. The answer displayed at the end of execution is the same one you would have obtained by pressing the keys one at a time manually. No prior programming experience is necessary to learn HP-34C programming.

Why Write Programs?

Programs are written to save time on repetitive calculations. Once you have written the keystroke procedure for a particular problem and recorded it in the calculator, you need no longer devote attention to the individual keystrokes that make up the procedure. You can let your HP-34C calculate the solution to each problem for you. And you can have more confidence in the answer. Why? Because once you have checked that your program is correctly recorded in the calculator, you may be sure that the calculator will execute your commands faithfully, without the slips you might make if you had to manually press the keys over and over again. The calculator performs the drudgery, leaving your mind free for more creative work.

Before proceeding, let's take another look at the powerful programming features designed into your HP-34C:

- An easily understood programming language.
- Twelve labels you can use (and re-use) to designate various programs and portions of programs.
- Fully merged program lines. Commands requiring multiple keystrokes—such as [f] [SIN] or [STO] [+] 1—consume only one line of program memory.

- Automatic Memory Allocation. Possible memory combinations range from 21 storage registers and 70 lines of programming to 1 storage register (the I-register) and 210 lines of programming. Memory conversion occurs at the rate of seven lines of programming for each data storage register—automatically!
- Decision-making capability for more sophisticated routines.
- Easy to use editing features for correcting and modifying programs.
- Six levels of subroutines and four flags to help simplify otherwise complicated programs.
- Indirect storage, recall, branching, and subroutine calls to automatically control data, decisions, and program control.
- Increment/decrement counter and looping control.

Together, these features provide you with the tools necessary to tackle complex problems with confidence.

Three Calculator Modes

Your HP-34C calculator has three operational modes:

1. Manual run mode.
2. Program mode.
3. Automatic run mode.

Manual Run Mode. The functions and operations you have learned about in the first part of this handbook and in *Solving Problems With Your Hewlett-Packard Calculator* are performed manually one at a time. These functions combined with the automatic memory stack enable you to calculate with ease.

Program Mode. In program mode the functions and operations you have learned about are not executed but instead are recorded, in a part of the calculator called program memory, for later execution. To get into program mode, simply slide the PRGM-RUN switch to PRGM ▥█ . All operations on the keyboard except the following can be recorded for later execution when the calculator is in program mode.

These operations cannot be recorded:

f CLEAR PRGM	f CLEAR PREFIX	g MEM
h SST	GTO • *nnn*	h MANT
h BST	h DEL	STO ENTER↑

You will find all of the above operations except h MANT and STO ENTER↑ useful when keying in and editing your programs.*

Automatic Run Mode. As you have learned, the HP-34C will automatically execute a list of operations when the calculator is in run mode if they have previously been recorded in program memory. Instead of pressing each key manually, the recorded operations are executed sequentially in automatic run mode. Typically, you press only one key to start the calculator at the beginning of the list. The entire list of recorded operations is then executed much more quickly than you could have executed them yourself.

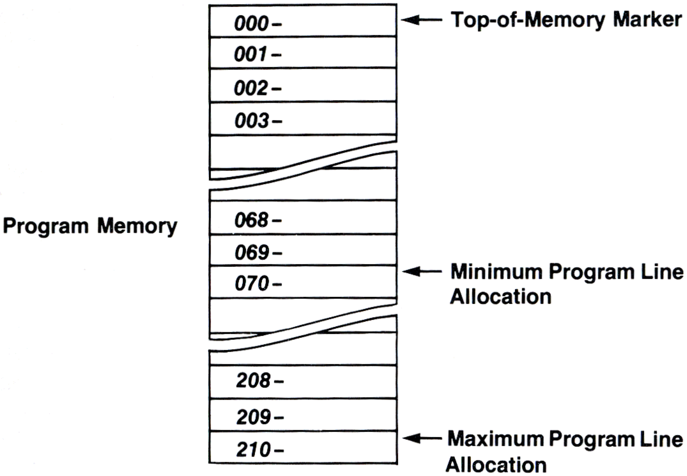
Looking at Program Memory

As you may remember from the program you created in section 1, the keystrokes used to calculate a solution manually are also used when you write a program to calculate the solution automatically. These keystrokes are stored in the calculator's program memory. When you slide the PRGM-RUN switch to PRGM you can examine the contents of program memory, one line at a time. Press GTO • 000 to return the calculator to the beginning of program memory. If you have not already done so, slide the PRGM-RUN switch to PRGM. The display should show ***000–***.

Program memory consists of from 70 to 210 lines, together with a top-of-memory marker which is the ***000–*** you now see in your display. Program memory operates separately from the stack, LAST X, I, and available storage registers.†

* Pressing h MANT does nothing in *program* mode, but pressing STO ENTER↑ in either program or run mode will perform the self-check as instructed, and will clear the stack, LAST X register, and flags, reset trig mode to degrees, and reset the calculator to line 000 in program memory.

† "Available" storage registers refers to data storage registers that are not converted to program memory.



With the PRGM-RUN switch set to PRGM ▥■, the number that you see on the left side of the display indicates the line number of program memory to which the calculator is set. Press [f] CLEAR [PRGM], [h] [LBL] [A], the first keystrokes of the Moon Surface Area program (refer to page 17), and the display will change to:

$$\text{Line number} \longrightarrow \underbrace{001}–25,\ 13,\ 11$$

The calculator is now set to line 001 of program memory, as indicated by the number ***001*** that you see on the left side of the display. The other numbers in the display are keycodes for the keystrokes that have been loaded into that line of program memory. Press [g] [x^2]. Your display shows:

002–   15   3

The number 002 on the left side of the display indicates that you are now at line two of the program.

Each line of program memory can "remember" a single instruction, whether that instruction consists of one, two, or three keystrokes. Thus, one line of program memory might contain a single-keystroke instruction like [CHS], while another line of program memory could contain the three-keystroke instruction [STO] [+] 6 (adds the value in the displayed X-register to the contents of register number 6).

But how do those numbers in the display relate to the actual keystrokes of program commands? This question brings us to the next step in mastering your HP-34C—keycodes.

Keycodes

Let's take another look at the program instructions we just entered. Press $\boxed{\text{h}}$ $\boxed{\text{BST}}$. Your display will now show the first line of the Moon Surface Area program:

$$\textbf{Line number} \longrightarrow \underbrace{\mathit{001}}-\underbrace{\mathit{25, 13, 11}} \longleftarrow \textbf{Keycodes}$$

As you know, the number code ***001*** appearing on the left side of the display designates the line number of program memory. The next digit pair, ***25***, represents the $\boxed{\text{h}}$ keystroke; ***13***, the $\boxed{\text{LBL}}$ keystroke; and ***11***, the $\boxed{\text{A}}$ keystroke. The first digit of each pair denotes the row the key is located in; the second digit denotes the number of the key in the row. So ***25*** tells you that the key is in the second row on the calculator and that it is the fifth key in that row, or, the $\boxed{\text{h}}$ key. In this manner each key on the keyboard is represented by a two-digit keycode, except for the digit keys zero through nine. For convenience, these keys and their respective alternate functions are coded 0 through 9. Let's see an example. Press $\boxed{\text{h}}$ $\boxed{\text{SST}}$ once. Your HP-34C's display will now show the second line of the Moon Surface Area program, $\boxed{\text{g}}$ $\boxed{x^2}$:

$$\textbf{Line number} \longrightarrow \underbrace{\mathit{002}}- \quad \underbrace{\mathit{15} \quad \mathit{3}} \longleftarrow \textbf{Keycodes}$$

From the above, we know that ***002*** is the program line number and ***15*** is the first row, fifth key, or the $\boxed{\text{g}}$ key. Because the $\boxed{\text{g}}$ prefix key is part of this instruction, the ***3*** denotes the x^2 function which is located on the 3 digit key. In calculator jargon, $\boxed{\text{g}}$ $\boxed{x^2}$ is a "shifted function" of the 3 key, just as the asterisk is a shifted function of a typewriter key.

The remaining keystrokes for the Moon Surface Area program are shown below with their corresponding displays. Press each key in turn and verify the keycodes shown in the display.

Keystrokes	**Display**
$\boxed{\text{h}}$ $\boxed{\pi}$	***003- 25 73***
$\boxed{\times}$	***004- 61***
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	***005- 25 12***

In this case, a program consisting of 10 keystrokes takes only five lines of program memory.

Problems

1. What would be the keycodes for the following operations:

   $\boxed{\text{h}}$ $\boxed{1/x}$, $\boxed{\text{g}}$ $\boxed{\text{GRD}}$, $\boxed{\text{f}}$ $\boxed{\to\text{H.MS}}$, $\boxed{\text{STO}}$ $\boxed{+}$ 1? (Answers: ***25 2; 15 13; 14 6; 23, 51, 1***).

2. How many lines of program memory would be required to load the following sections of programs?

   a. 2 $\boxed{\text{ENTER}\uparrow}$ 3 $\boxed{+}$.

   b. 10 $\boxed{\text{STO}}$ 6 $\boxed{\text{RCL}}$ 6 $\boxed{\times}$.

   c. 100 $\boxed{\text{STO}}$ 1 50 $\boxed{\text{STO}}$ $\boxed{\times}$ 1 $\boxed{\text{RCL}}$ 2 $\boxed{\text{h}}$ $\boxed{\pi}$ $\boxed{\times}$.

   (Answers: a, 4; b, 5; c, 10.)

Clearing a Program

The Continuous Memory feature of your calculator preserves any programs loaded into program memory even while the calculator is turned off. To clear program memory, turn the calculator on, slide the PRGM-RUN switch to PRGM ▥■, and press $\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$. All lines of program memory formerly occupied by programs you cleared using $\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$ are again available for storing new program instructions. If the programs you cleared occupied more than 70 lines of program memory, the lines the programs used in excess of the first 70 are automatically reallocated to data storage registers. Note that if you press $\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$ in RUN mode, the calculator resets to line 000, but program memory is *not* cleared.

If power to the calculator is interrupted (that is, battery failure), all instructions in program memory and all data in the storage registers may be lost. When power is restored and the unit turned on, ***Pr Error*** appears in the display to warn of this loss.

Creating Your Own Program

In Meet the HP-34C, at the beginning of this handbook, we created a program that calculated the surface area of a sphere, given the diameter of that sphere. Now let's create another program to show you how to use some of the other features of the HP-34C.

If you wanted to use your HP-34C to calculate manually the area of a circle using the formula $A = \pi r^2$ you could first key in the radius r, then square it by pressing [g] [x^2]. Next you would summon π into the display by pressing [h] [π]. Finally you would multiply the squared radius and π together by pressing [×].

Remember that a *program* to calculate a given solution is little more than the keystrokes you would use to calculate that solution manually. Thus, to create an HP-34C program for calculating the area of any circle, you will want to identify the keystroke sequence used to calculate the area of a circle manually.

The keystroke sequence for calculating the area of a circle according to the formula $A = \pi r^2$ are:

[g] [x^2]
[h] [π]
[×]

You will load into program memory these keystrokes plus, normally, two more operations, [h] [LBL] [A] and [h] [RTN]. [h] [LBL] [A] is called a label address and is used to begin the program. [h] [RTN] is used to end the program.

Beginning a Program

To define the beginning of a program use an [h] [LBL] *(label)* instruction followed by one of the letter keys ([A] or [B]), or by one of the digit keys (0 through 9). The use of labels permits you to have several different programs or parts of programs loaded into the calculator at any time, and to run them in the order you choose.

Ending a Program

To define the end of a program, you can use an [h] [RTN] *(return)* instruction. When the calculator encounters a [RTN] instruction while executing a program, it immediately transfers execution to line 000 and halts (unless executed as part of a subroutine—more about subroutines later).

Note: When a running program encounters the end of *occupied* program memory, the effect is the same as if an h RTN had been encountered. This means that when programming, if your *last* instruction in occupied program memory would be an h RTN, it can be eliminated, saving you one line of memory space.

If you want a program to halt at a certain line in memory without returning to line 000, you can key in a R/S instruction at that line. When a running program encounters a R/S instruction in program memory, execution simply halts. If you switch from RUN to PRGM mode, you will see the next line of program memory after the R/S instruction. (Remember that the calculator returns to line 000 and halts after executing the last instruction in program memory whenever the last instruction is any command other than R/S, GSB, GTO, or a RTN from a subroutine; so there is normally no need to put R/S at the end of the last program in memory to halt execution.)

The complete program to calculate the area of any circle given its radius is:

h LBL A	Assigns name to and defines beginning of program.
g x^2	Squares the radius.
h π	Summons π into the display.
×	Multiplies r^2 by π and displays the answer.
h RTN	Defines the end of and stops the program.

Loading a Program

When the calculator is set to PRGM, the functions and operations that are normally executed when you press the keys are not executed. Instead, they are stored in program memory for later execution. *All keyboard operations except the nine listed on page 46 can be loaded into program memory for later execution.*

To prepare for loading a complete program into the calculator:

1. Slide the PRGM-RUN switch to PRGM.
2. Press f CLEAR PRGM to clear program memory of any previous programs.

You can tell that the calculator is at the top of program memory because the digits 000 appear at the left of the display. The digits appearing at the left of the display when the calculator is in PRGM mode always indicate the program memory line number being shown at the time.

The keys you press to load the program calculating the area of a circle are:

[h] [LBL] [A]
[g] [x^2]
[h] [π]
[×]
[h] [RTN]

Press the first key, [h], of the program.

Keystrokes	Display
[h]	*000–*

You can see that the display of program memory has not changed. Now press the second and third keys of the program.

Keystrokes	Display
[LBL]	*000–*
[A]	*001–25, 13, 11*

When a new program memory line number appears on the left of the display, it indicates that a complete operation has been loaded into that line. As you can see from the keycodes present on the right side of the display, the complete operation is [h](keycode ***25***), [LBL](keycode ***13***), [A](keycode ***11***). Nothing is loaded into program memory until a complete operation (whether one, two, or three keystrokes) has been specified.

Now load the remainder of the program by pressing the following keys. Observe the program memory line numbers and keycodes.

Keystrokes	Display
[g] [x^2]	*002– 15 3*
[h] [π]	*003– 25 73*
[×]	*004– 61*
[h] [RTN]	*005– 25 12*

The program for solving the area of a circle given its radius is now loaded into your HP-34C's program memory. Notice that nothing could be loaded into the top-of-memory marker, line 000.

Running a Program

Programs are executed in automatic run mode. With the PRGM-RUN switch in ■▯RUN position, key in any data that is required, and press the letter key ($\boxed{A}$ or $\boxed{B}$), that labels your program.

For example, to use the program now in the calculator to calculate areas of circles with radii of 3 inches, 6 meters, and 9 miles:

First, slide the PRGM-RUN switch to ■▯RUN .

Keystrokes	Display	
3 $\boxed{A}$	***28.2743***	Square inches.
6 $\boxed{A}$	***113.0973***	Square meters.
9 $\boxed{A}$	***254.4690***	Square miles.

Now let's see how the HP-34C executed this program.

Searching for a Label

When you switched the PRGM-RUN switch to RUN, the calculator was set at line 005 of program memory, the last line you had filled with an instruction when you were loading the program. When you pressed the $\boxed{A}$ key, the calculator began *searching* sequentially downward through program memory, beginning with line 005, for a $\boxed{LBL}$ $\boxed{A}$ instruction. When the calculator searches, it does not execute instructions.

Because line 005 did not contain the $\boxed{h}$ $\boxed{LBL}$ $\boxed{A}$ instruction, and no further lines of program memory were occupied, your HP-34C returned to line 000 and resumed searching downward through program memory. When the calculator found the $\boxed{h}$ $\boxed{LBL}$ $\boxed{A}$ instruction in line 001 it then began *executing* your program.

Executing Instructions

The calculator executes instructions in exactly the order you keyed them in, performing the $\boxed{\text{g}}$ $\boxed{x^2}$ operation in line 002 first, then $\boxed{\text{h}}$ $\boxed{\pi}$ in line 003, etc., until it executes an $\boxed{\text{h}}$ $\boxed{\text{RTN}}$ instruction, a $\boxed{\text{R/S}}$ *(run/stop)* instruction, or encounters the end of occupied program memory. Since there is an $\boxed{\text{h}}$ $\boxed{\text{RTN}}$ instruction in line 005, execution returns to line 000 and halts. The calculator then displays the contents of the X-register.

It is normally best to use $\boxed{\text{A}}$ or $\boxed{\text{B}}$ to define the beginning of a program and to save 0 through 9 for subroutine labels (more on subroutines later). Why? Labels $\boxed{\text{A}}$ and $\boxed{\text{B}}$ require only one keystroke to begin execution, as in our area of a circle program. But if you have several short programs to key into your HP-34C you can use labels 0 through 9 to address some of the individual programs. Using numerical labels requires an additional keystroke, $\boxed{\text{GSB}}$, for program execution. To illustrate, let's load and execute our area of a circle program using 0 for the label.

Slide the PRGM-RUN switch to PRGM ▮.

Keystrokes	Display
$\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$	***000–***
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 0	***001–25, 13, 0***
$\boxed{\text{g}}$ $\boxed{x^2}$	***002– 15 3***
$\boxed{\text{h}}$ $\boxed{\pi}$	***003– 25 73***
$\boxed{\times}$	***004– 61***
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	***005– 25 12***

Slide the PRGM-RUN switch back to ▮ RUN .

Now, execute the program using the example from page 53. This time, because of our label change, press $\boxed{\text{GSB}}$ 0 instead of $\boxed{\text{A}}$.

Keystrokes	Display
3 $\boxed{\text{GSB}}$ 0	***28.2743***
6 $\boxed{\text{GSB}}$ 0	***113.0973***
9 $\boxed{\text{GSB}}$ 0	***254.4690***

If you try to execute a label (LBL) that is not contained as an instruction in program memory, the HP-34C will display ***Error 4***. For example, if your calculator contains only the program for area of a circle that you just keyed in, you can cause an Error 4 condition by simply pressing a letter key.

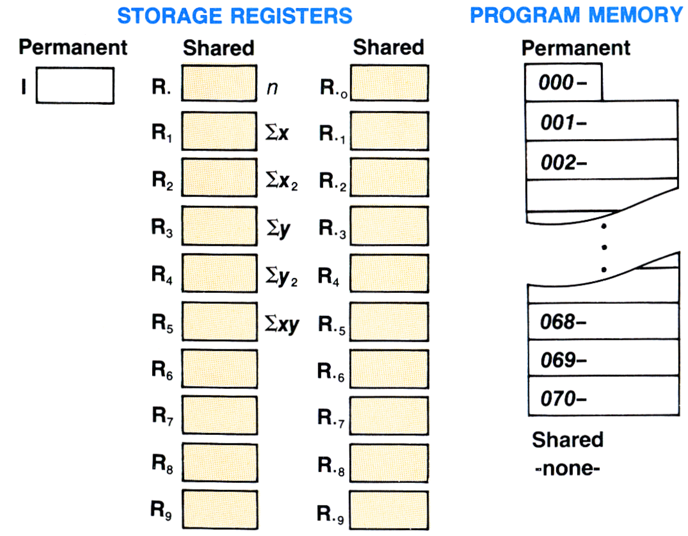
Keystrokes	**Display**
B	***Error 4***

To clear the error message from the display, press CLX or any other key. The calculator remains set at the current line of program memory.

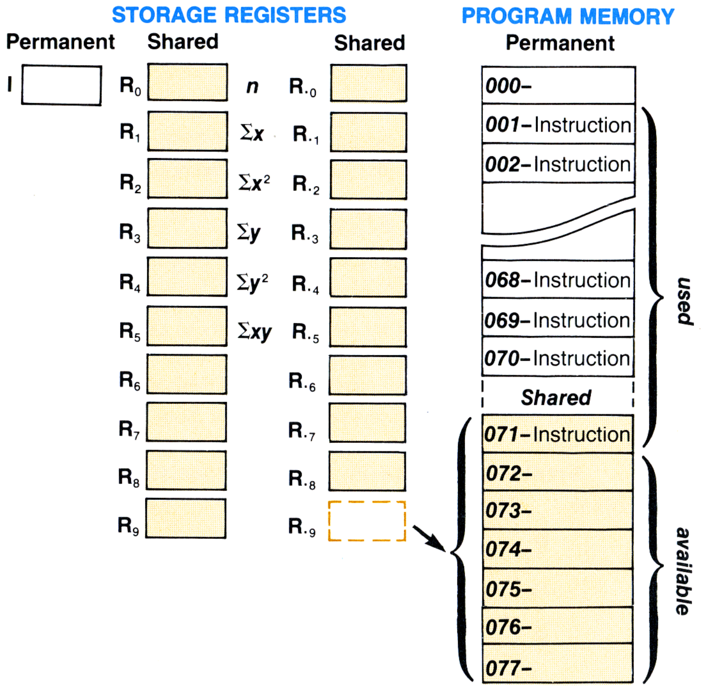
Automatic Memory Allocation

Converting Storage Registers to Program Memory

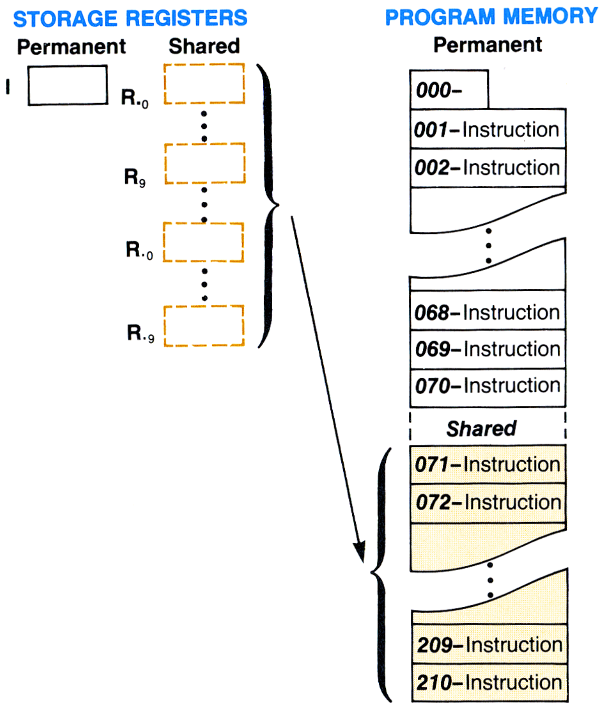
The automatic memory allocation designed into your HP-34C gives you increased versatility by converting storage registers to lines of program memory only as needed. You begin programming with 70 lines of program memory and 20 storage registers (plus the I-register, described in section 7). With 0 to 70 instructions in program memory, the allocation looks like this:



When you key in the 71st line of programming, storage register $R_{.9}$ converts to 7 lines of additional program memory. Now the memory allocation looks like this:



When you record a full 210 lines of program memory, the calculator's memory registers look like this:



Program memory is separate from the four stack registers and the LAST X register. Notice that instead of the original 21 storage registers (R_0 through R_9, $R_{.0}$ through $R_{.9}$, and I) we now have just the non-convertable I-register. What happened to storage registers R_0 through R_9 and $R_{.0}$ through $R_{.9}$? They were converted to program memory at the rate of seven lines per register. The following table shows the allocation of the lines of program memory to their respective storage registers.

$R_{.9}$	071—077	R_9	141—147
$R_{.8}$	078—084	R_8	148—154
$R_{.7}$	085—091	R_7	155—161
$R_{.6}$	092—098	R_6	162—168
$R_{.5}$	099—105	R_5	169—175
$R_{.4}$	106—112	R_4	176—182
$R_{.3}$	113—119	R_3	183—189
$R_{.2}$	120—126	R_2	190—196
$R_{.1}$	127—133	R_1	197—203
$R_{.0}$	134—140	R_0	204—210

When all 210 lines of program memory are occupied, attempting to insert an additional program instruction anywhere in memory results only in ***Error 4*** appearing in the display. The additional instruction will be ignored and none of the original 210 lines will be lost.

As you can see, each time currently available programming space is filled, keying in another command automatically converts the next remaining storage register to seven more lines of program memory. For example, filling the first 77 lines and then keying a command into line 78 converts register $R_{.8}$ to 7 more lines of program memory (lines 78-84), and so on.

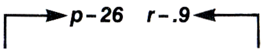
Note: Your HP-34C converts storage registers to program lines in reverse numerical order, from $R_{.9}$ to $R_{.0}$ and then from R_9 to R_0. For this reason it is good practice to program your STO and RCL operations using data registers in the opposite order; that is, beginning with register R_0. This procedure helps avoid accidentally programming STO and RCL for data registers which have been converted to lines of program memory. Remember also that the calculator does not retain data previously stored in registers that are later converted to lines of program memory.

Converting Program Memory to Storage Registers

Pressing $\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$ in PRGM mode converts all shared program memory (lines 071-210) to storage registers R_0 through $R_{.9}$. However, deleting individual lines of program memory allows you to convert portions of shared memory to storage registers without clearing all of program memory. (More on deleting lines of memory in section 4, Editing.)

Using $\boxed{\text{MEM}}$

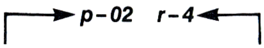
The $\boxed{\text{MEM}}$ *(memory)* function on your calculator describes the current memory allocation in or out of program mode. When you press $\boxed{\text{g}}$ $\boxed{\text{MEM}}$ the display shows both (1) the number of currently unused *(available)* program lines you must load before a storage register will be converted, and (2), the name of the storage register which is next in line to be converted ($R_{.9}$ through $R_{.0}$, R_9 through R_0). For example, if you press $\boxed{\text{g}}$ $\boxed{\text{MEM}}$ with 44 lines of program memory occupied, you will see the following display:



Lines remaining to be occupied before the calculator automatically converts a storage register to 7 more program lines.

The next storage register to be converted.

If you press $\boxed{\text{g}}$ $\boxed{\text{MEM}}$ with 173 lines of program memory occupied, you will see this display:



Lines remaining to be occupied before the calculator automatically converts a storage register to 7 more program lines.

The next storage register to be converted.

If you press g MEM with 205 lines of program memory occupied, you will see this display:

```
p-05  r-
```

Lines remaining to be occupied before *all* lines of program memory are occupied.

No more storage registers can be converted to program memory.

As long as you are pressing MEM, the memory allocation will be displayed. When you release the MEM key, the calculator returns to the original display. So at any time, you can find out the number of lines available for programming and the number of registers available for storing data. Because the I-register is a permanent storage register with special functions, it is not covered by the MEM operation.

> **Note:** Remember that the statistical functions involve registers R_0 through R_5. If one or more of these last six registers are converted to lines of program memory, attempts to execute statistical functions will result in an ***Error 2*** display.

Writing a Third Program

To further explore the programming capabilities of your HP-34C let's write a third program. Suppose you want to write a program that will calculate the increase in volume of a spherical balloon as its diameter increases using the formula:

$$\text{Increase in volume} = {}^1\!/_6\pi\,(d_1^3 - d_0^3),$$

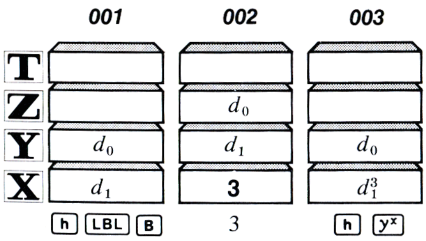
where d_0 is the original diameter of the balloon and d_1 is the new diameter. If d_0 were entered in the Y-register and d_1 were keyed into the X-register, the problem could be solved manually by pressing the keys shown in the left-hand column below. The program keystrokes for this problem are the same as the manual keystrokes. Switch the PRGM-RUN switch to PRGM and press the keys shown below.

Keystrokes	Display	
f CLEAR PRGM	000-	
h LBL B	001-25, 13, 12	
3	002- 3	
h y^x	003- 25 3	Cube the new diameter.
$x \rightleftarrows y$	004- 21	
3	005- 3	
h y^x	006- 25 3	Cube the original diameter.
−	007- 41	Subtract the cubes.
h π	008- 25 73	
×	009- 61	Multiply by π.
6	010- 6	
÷	011- 71	Divide by 6.
h RTN	012- 25 12	

Slide the PRGM-RUN switch to RUN.

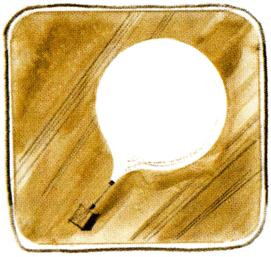
Notice that an ENTER↑ command was not included to separate the number 3 in line 002 from the digits you will key in later. Including ENTER↑ after the LBL instruction would not cause an error in this example, but is not necessary. Why? When a running program executes a LBL instruction, *stack lift is enabled*. When a new number is then entered into the X-register the stack automatically lifts. Here is how this works when you run the above program with d_0 entered into the Y-register and d_1 keyed into the X-register.

Stack Registers



If you are unsure how other operations affect the stack, see appendix E, Stack Lift and LAST X.

Example: Find the increase in volume of a spherical balloon if the diameter changes from 30 feet to 35 feet.



Keystrokes	Display	
30 [ENTER↑]	***30.0000***	Enter original diameter into Y.
35 [B]	***8,312.1306***	Key new diameter into X and run the program. The answer is displayed in cubic feet.

Program Stops and Pauses

When programming, there may be occasions when you want a program to halt during execution so that you can key in data. Or you may want the program to pause so that you can quickly view results before the program automatically resumes running. Two keys, [R/S] *(run/stop)* and [PSE] *(pause)*, are used for program interruptions.

Planned Stops During Program Execution

The [R/S] *(run/stop)* function can be used either as an instruction in a program or as an operation pressed from the keyboard.

When pressed from the keyboard:

1. If a program is running, R/S halts program execution.

2. If a program is stopped or not running, and the calculator is in RUN mode, pressing R/S starts the program running. Execution then begins with the first line of program memory following the R/S instruction. (When R/S is pressed and held in RUN mode, it displays the line number and keycode of that current line—when released, execution begins with that line.)

You can use these features of the R/S instruction to stop a running program at points where you want to key in data. After the data has been keyed in, restart the program using the R/S key from the keyboard.

Example: Universal Tins, a canning company, needs to calculate the volumes of various cylindrically-shaped cans. Universal would also like to be able to record the area of the base of each can before the volume is calculated.

The following program calculates the area of the base of each can and then stops. After you have written down the result, the program can be restarted to calculate the final volume. The formula used is:

$$\text{Volume} = \text{base area} \times \text{height} = \pi r^2 \times h$$

The radius (r) and the height (h) of the can are keyed into the X- and Y-registers, respectively, before the program is run.

To record this program, set the PRGM-RUN switch to PRGM ▥■, then key in the following program instructions.

Keystrokes	Display	
f CLEAR PRGM	*000-*	Clears program memory and displays line 000.
h LBL A	*001-25, 13, 11*	
g x^2	*002- 15 3*	Square the radius.
h π	*003- 25 73*	Place π in X.
×	*004- 61*	Calculate the area of the base.
R/S	*005- 74*	Stop to record the area.
×	*006- 61*	Calculate the final volume.
h RTN	*007- 25 12*	

Set the PRGM-RUN switch to ■▥ RUN. Then use the program to complete the table below:

Height	Radius	Area of Base	Volume
25	10.0	?	?
8	4.5	?	?

Keystrokes	Display	
25 ENTER↑	*25.0000*	Enter the height into the Y-register.
10 A	*314.1593*	Key the radius into the X-register and calculate area. Program stops to display the area.
R/S	*7,853.9816*	Volume of first can is calculated.
8 ENTER↑	*8.0000*	Enter the height into the Y-register.
4.5 A	*63.6173*	Key the radius into the X-register and calculate area. Program stops to display the area.
R/S	*508.9380*	Second volume is calculated.

With the height in the Y-register and the radius in the X-register, pressing A in automatic RUN mode calculates the area of the can's base; the program stops at the first R/S instruction encountered. Pressing R/S calculates the volume of the can. Program execution then returns to line 000 and halts.

Pausing During Program Execution

An h PSE instruction executed in a program interrupts program execution to display results momentarily before execution is resumed. The length of the pause is about 1 second, but you can use more than one consecutive h PSE instruction to lengthen the time.

To see how h PSE can be used in a program, we'll modify the cylinder volume program in the previous example. In the new program the area of the base will be briefly displayed before the volume is calculated. This example will also show how different programming approaches can be taken to solve the same problem.

To key in the program, set the PRGM-RUN switch to PRGM . Press f CLEAR PRGM to clear program memory and display line 000. Then key in the following program instructions.

Keystrokes	**Display**	
f CLEAR PRGM	*000–*	
h LBL A	*001–25, 13, 11*	
g x^2	*002– 15 3*	Squares the radius in X.
h π	*003– 25 73*	Places π in X.
×	*004– 61*	Calculates the area of the base.
h PSE	*005– 25 74*	Pauses to show the base area for one second.
×	*006– 61*	Calculates final volume of can.
h RTN	*007– 25 12*	

This program also assumes the height has been entered into the Y-register and the radius has been keyed into the X-register. If you have stored the instructions, set the PRGM-RUN switch to ▮▯ RUN . Now complete the table below using the new program.

Height	Radius	Area of Base	Volume
20	15	?	?
10	5	?	?

Keystrokes	Display	
20 $\boxed{\text{ENTER}\uparrow}$	***20.0000***	Enter the height into the Y-register.
15 $\boxed{\text{A}}$	***706.8583***	Key the radius into the X-register and calculate. Area of base is displayed for 1 second.
	14,137.1669	Program stops, displaying the volume.
10 $\boxed{\text{ENTER}\uparrow}$	***10.0000***	Enter the second height into Y.
5 $\boxed{\text{A}}$	***78.5398***	Key the radius into the X-register and calculate. Area of base is displayed for 1 second.
	785.3982	Program stops, displaying the volume.

Unexpected Program Stops

At times a mistake of some kind in your program will stop program execution. To help you determine why the calculator stopped in the middle of a program, possible reasons are listed below.

Executing $\boxed{\text{h}}$ $\boxed{\text{RTN}}$. Unless in a subroutine, whenever $\boxed{\text{h}}$ $\boxed{\text{RTN}}$ is executed in a program, the calculator immediately returns to line 000 and halts.

Encountering the End of Program Memory. When the final instruction in program memory is not GTO, GSB, RTN or R/S, and is not in a subroutine, a running program will encounter the end of occupied program memory, transfer immediately to line 000, and halt.

Pressing Any Key. Pressing any key halts program execution. Be careful to avoid pressing keys during program execution. The calculator has been designed so that program execution will *not* halt in the middle of a digit entry sequence. If you press any key while a number is being placed in the X-register by a running program, the entire number will be ''written'' and the following line will be executed by the program before the calculator halts.

When a program is halted, you can resume execution by pressing R/S from the keyboard in RUN mode. When you press R/S, the program resumes execution where it left off as though it had never stopped at all.

Error Stops. If the calculator attempts to execute any error-causing operation (refer to appendix D, Error Indications) during a running program, execution immediately halts and the calculator displays the word ***Error*** and a number. To see the line number and keycode of the error-causing instruction, you can switch the calculator to PRGM mode.

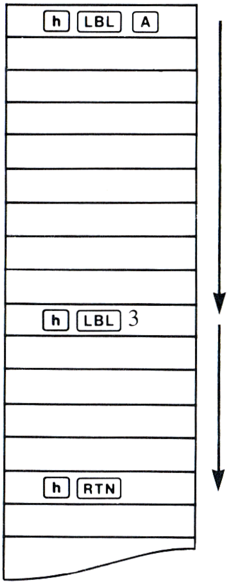
Overflow Calculations. Your HP-34C has been designed so that by looking at the display you can always tell why the calculator stops. If program execution stops because the result of a calculation in the X-register is a number with a magnitude greater than $9.999999999 \times 10^{99}$, all 9's are displayed with appropriate sign. It is then easy to determine the operation that caused the overflow by switching to PRGM mode and identifying the keycode in the display.

If an attempted storage register arithmetic operation would result in overflow in a storage register, the calculator halts and displays ***Error 1***. The number in the affected storage register remains unchanged from its previous value. When you clear the error message, the last number in the display returns.

If the result of a calculation is a number with a magnitude less than $1.000000000 \times 10^{-99}$, zero will be substituted for that number and a running program will continue to execute normally. This is known as an underflow.

Labels

The labels ([A], [B], 0-9) in your programs act as addresses—they tell the calculator where to begin or resume execution. When a label is encountered as part of a program, execution merely "falls through" the label and continues onward. For example, in the program segment shown below, if you press [A], execution would begin at [h] [LBL] [A] and continue downward through program memory, on through the [h] [LBL] 3 instruction, until the [RTN] was encountered and execution returned to line 000 and halted.



When you press [A] ... execution begins here.

No [h] [RTN] here ... so execution falls through the [h] [LBL] 3 instruction ...

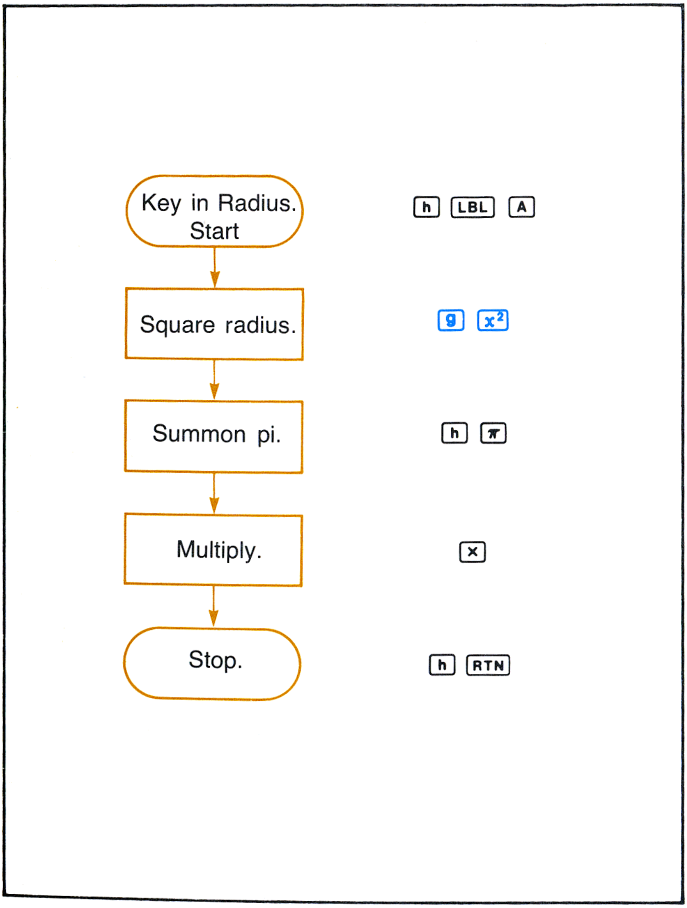
... and continues to the [RTN], then transfers to line 000 and halts.

Flowcharts

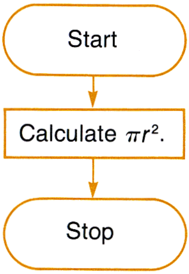
At this point, we digress for a moment from our discussion of the calculator itself to discuss a fundamental and extremely useful tool in programming—the flowchart.

A flowchart is an *outline* of the way a program solves a problem. With 210 possible instructions, it is quite easy to get "lost" while creating a long program, especially if you try to simply load the complete program from beginning to end with no breaks. A flowchart is a shorthand that can help you design your program by breaking it down into smaller groups of instructions. It is also very useful as documentation—a road map that summarizes the operation of a program.

A flowchart can be as simple or as detailed as you like. Here is a flowchart that shows the operations you executed to calculate the area of a circle according to the formula $A = \pi r^2$. Compare the flowchart to the actual instructions for the program:



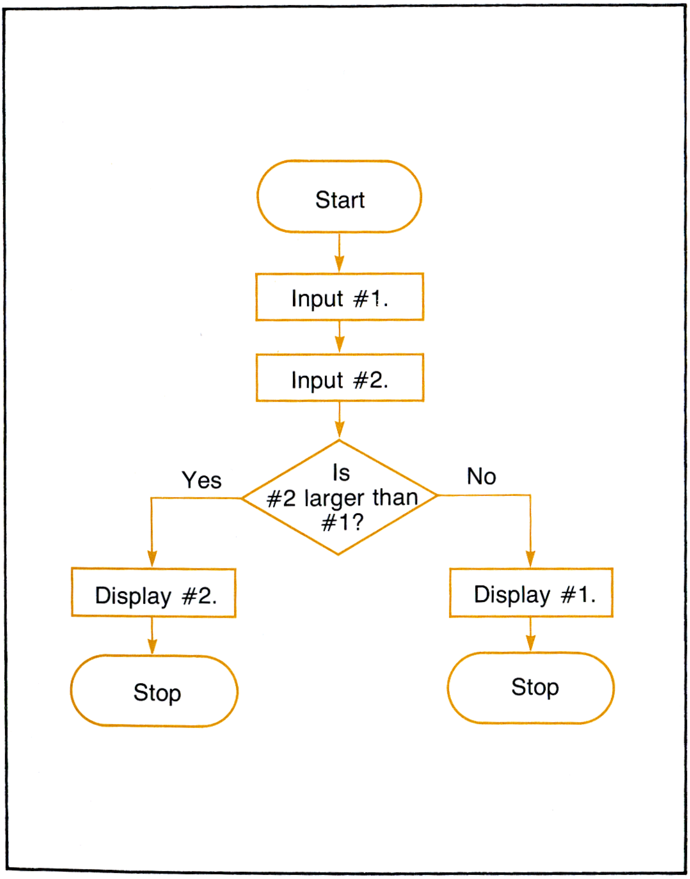
You can see the similarities. At times, a flowchart may duplicate the set of instructions exactly, as shown above. At other times, it may be more useful to have an entire group of instructions represented by a single block in the flowchart. For example, here is another flowchart for the program that calculates the area of a circle:



Here an entire group of instructions was replaced by one block in the flowchart. This is a common practice, and one that makes a flowchart extremely useful in visualizing a complete program.

You can see how a flowchart is drawn linearly, from the top of the page to the bottom. This represents the general flow of the program, from beginning to end. Although flowcharting symbols sometimes vary, throughout this handbook we have held to the convention of ovals for the beginning and end of a program or subroutine, and rectangles to represent groups of functions that take an input, process it, and yield a single output. We have used a diamond to represent a *decision,* where a single input can yield either of two outputs.

For example, if you had two numbers and wished to write a program that would display only the larger, you might design your program by first drawing a flowchart that looks like this:



After drawing the flowchart, you would go back and substitute groups of instructions for each element of the flowchart. When the program was loaded into the calculator and run, if #2 was larger than #1, the answer to the question "Is #2 larger than #1?" would be YES, and the program would take the left-hand path, display #2, and stop. If the answer to the question was NO, the program would execute the right-hand path, and #1 would be displayed. You will see later the many decision-making instructions available on your HP-34C.

As you work through this handbook, you will become more familiar with flowcharts. Use the flowcharts that illustrate the examples and problems to help you understand the many features of the calculator, or draw your own flowcharts to help you create, edit, eliminate errors in, and document your programs.

Problems

Here are four programming examples for you to try using material we've already covered. Possible solutions for these examples are shown on the following pages. However, you will receive the most benefit from the exercises by coming up with your own solutions before finding out how we've done it. Remember, there is usually more than one way to solve a programming problem. Perhaps you can improve on our solutions!

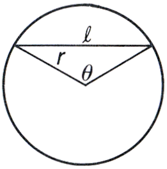
1. You have seen how to write, load, and run a program to calculate the area of a circle from its radius. Now write and load a program that will calculate the radius r of a circle given its area A using the formula $r = \sqrt{A/\pi}$. Be sure to slide the PRGM-RUN switch to PRGM and press $\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$ first to clear program memory. Define the program with $\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$ and $\boxed{\text{h}}$ $\boxed{\text{RTN}}$. After you have loaded the program, run it to calculate the radii of circles with areas of 28.2743 square inches, 113.0973 square meters, and 254.4690 square miles.

   (Answers: 3.0000 inches, 6.0000 meters, 9.0000 miles.)

2. Create a program to calculate the length of a chord ℓ subtended by angle θ on a circle of r radius using the equation $\ell = 2r \text{ Sin } \frac{\theta}{2}$.

Define this new program with [h] [LBL] [B] and use it to complete the following table:

r (meters)	θ	ℓ
25	30	?
50	45	?
100	90	?



Design your program for a r, θ order of data entry.

(Answers: 12.9410 meters, 38.2683 meters, 141.4214 meters.)

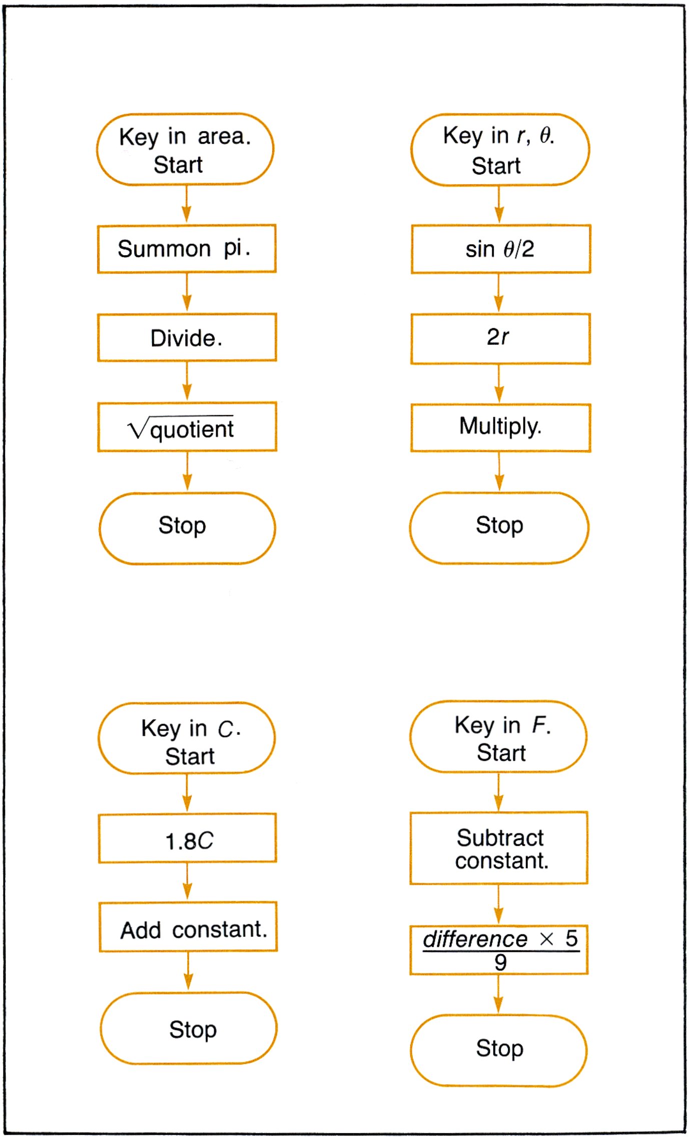
If you have difficulty programming for this example, go back to page 60 and study Writing a Third Program.

3. Write and load a program that will convert temperature in degrees Celsius to Fahrenheit, according to the formula $F = 1.8C + 32$. Define the program with [h] [LBL] 0 and [h] [RTN] and run it to convert Celsius temperatures of –40°, 0°, and +72°.

   (Answers: –40.0000°F, 32.0000°F, 161.6000°F.)

4. Create a program that will convert temperature in Fahrenheit back to Celsius according to the formula $C = (F - 32)5/9$. Define the program using [h] [LBL] 1 and [h] [RTN]. Run this new program to convert the temperatures in Fahrenheit you obtained back to Celsius.

If you wrote and loaded the programs as called for in problems 3 and 4, you should now be able to convert any temperature in Celsius to Fahrenheit by pressing [GSB] 0 and any temperature in Fahrenheit to Celsius by pressing [GSB] 1. Questions? Review Executing Instructions, beginning on page 54, concerning the use of [GSB] and labels 0 through 9 for addressing individual programs.


Key in area. Start
Summon pi.
Divide.
$\sqrt{\text{quotient}}$
Stop
Key in r, θ. Start
sin $\theta/2$
$2r$
Multiply.
Stop
Key in C. Start
$1.8C$
Add constant.
Stop
Key in F. Start
Subtract constant.
$\frac{\textit{difference} \times 5}{9}$
Stop

Example Problem Solutions

	Keystrokes	Display
Radius of a Circle	$\boxed{h}$ $\boxed{LBL}$ $\boxed{A}$	001–25, 13, 11
	$\boxed{h}$ $\boxed{\pi}$	002– 25 73
	$\boxed{\div}$	003– 71
	$\boxed{f}$ $\boxed{\sqrt{x}}$	004– 14 3
	$\boxed{h}$ $\boxed{RTN}$	005– 25 12
Length of a Chord	$\boxed{h}$ $\boxed{LBL}$ $\boxed{B}$	001–25, 13, 12
	2	002– 2
	$\boxed{\div}$	003– 71
	$\boxed{f}$ $\boxed{SIN}$	004– 14 7
	$\boxed{\times}$	005– 61
	2	006– 2
	$\boxed{\times}$	007– 61
	$\boxed{h}$ $\boxed{RTN}$	008– 25 12
Convert Celsius to Fahrenheit	$\boxed{h}$ $\boxed{LBL}$ 0	001–25, 13, 0
	1	002– 1
	$\boxed{\bullet}$	003– 73
	8	004– 8
	$\boxed{\times}$	005– 61
	3	006– 3
	2	007– 2
	$\boxed{+}$	008– 51
	$\boxed{h}$ $\boxed{RTN}$	009– 25 12
Convert Fahrenheit to Celsius	$\boxed{h}$ $\boxed{LBL}$ 1	001–25, 13, 1
	3	002– 3
	2	003– 2
	$\boxed{-}$	004– 41
	5	005– 5
	$\boxed{\times}$	006– 61
	9	007– 9
	$\boxed{\div}$	008– 71
	$\boxed{h}$ $\boxed{RTN}$	009– 25 12

Keystroke Solutions

1. Area $\boxed{A}$ = Radius
2. Radius $\boxed{ENTER\uparrow}$ θ $\boxed{B}$ = Length of Chord
3. C $\boxed{GSB}$ 0 = F
4. F $\boxed{GSB}$ 1 = C

Programming Techniques

The solutions to some types of problems require you to use the same variable several times during your calculations. As you may know, there is more than one way to program such solutions in your HP-34C. However, the program that is economical both in execution time and in program space is often the most desirable. Let's compare two different ways we can approach the solution to a problem using the same variable several times. For example, the polynomial $f(x) = x^4 + 3x^3 - x^2 + 4x - 1$ uses the variable x four times; i.e., x^4, x^3, x^2, x. This means that four powers of x will be needed to calculate $f(x)$. Your task is to write a program that both describes $f(x)$ mathematically *and* makes available a copy of the variable x each time it is needed during program execution. You can do this with reasonable efficiency in one of two ways. Either initially store a copy of the variable for later recall wherever it is needed; or, better, write your program so that the stack need only be filled with copies of the variable prior to execution. The advantages of this stack fill method over the storage method are:

1. Your program can easily be written to keep a copy of the variable either ready for immediate use or accessible with an [x⇄y] instruction. (Remember that each time the stack drops, the T-register duplicates the number which last occupied it before the stack dropped.) This means you use fewer lines of program memory because [STO] and [RCL] instructions are unnecessary.

2. A storage register is saved for other uses.

3. Stack fill is convenient for evaluating polynomial expressions generally, and for use with most [SOLVE] and [$\int_y^x$] applications (more on [SOLVE] and [$\int_y^x$] later).

Now let's look at a program that evaluates the expression $x^4 + 3x^3 - x^2 + 4x - 1$ using the stack fill method. This time we'll see just the program instructions and stack contents. Examine the program instructions line by line and be sure you understand how and why each instruction affects the stack. Assume that the value of x is already in the stack when program execution begins.

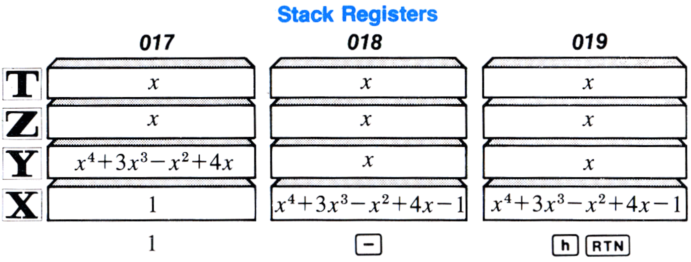
Stack Registers

	001	002	003	004	005
T	x	x	x	x	x
Z	x	x	x	x	x^4
Y	x	x	x	x^4	x
X	x	4	x^4	x	3
	$\boxed{h}\ \boxed{LBL}\ \boxed{A}$	4	$\boxed{h}\ \boxed{y^x}$	$\boxed{x \gtrless y}$	3

	006	007	008	009	010
T	x	x	x	x	x
Z	x	x^4	x	x	x
Y	x^4	x^3	x^4	x	x^4+3x^3
X	x^3	3	$3x^3$	x^4+3x^3	x
	$\boxed{h}\ \boxed{y^x}$	3	$\boxed{\times}$	$\boxed{+}$	$\boxed{x \gtrless y}$

	011	012	013
T	x	x	x
Z	x	x	x
Y	x^4+3x^3	x	$x^4+3x^3-x^2$
X	x^2	$x^4+3x^3-x^2$	x
	$\boxed{g}\ \boxed{x^2}$	$\boxed{-}$	$\boxed{x \gtrless y}$

	014	015	016
T	x	x	x
Z	$x^4+3x^3-x^2$	x	x
Y	x	$x^4+3x^3-x^2$	x
X	4	$4x$	$x^4+3x^3-x^2+4x$
	4	$\boxed{\times}$	$\boxed{+}$



Notice that extra copies of the variable remain in the stack after the program has been run. If, for any reason, you want to return a copy of the variable to the display after evaluating $f(x)$ at that variable, simply press $\boxed{x\gtrless y}$.

To experiment with the stack fill method, key in the program for evaluating the above expression and try running some examples.

Slide the PRGM-RUN switch to PRGM ▮.

Keystrokes	Display
$\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$	000–
$\boxed{\text{h}}\ \boxed{\text{LBL}}\ \boxed{\text{A}}$	001– 25, 13, 11
4	002– 4
$\boxed{\text{h}}\ \boxed{y^x}$	003– 25 3
$\boxed{x\gtrless y}$	004– 21
3	005– 3
$\boxed{\text{h}}\ \boxed{y^x}$	006– 25 3
3	007– 3
$\boxed{\times}$	008– 61
$\boxed{+}$	009– 51
$\boxed{x\gtrless y}$	010– 21
$\boxed{\text{g}}\ \boxed{x^2}$	011– 15 3
$\boxed{-}$	012– 41
$\boxed{x\gtrless y}$	013– 21
4	014– 4
$\boxed{\times}$	015– 61
$\boxed{+}$	016– 51
1	017– 1
$\boxed{-}$	018– 41
$\boxed{\text{h}}\ \boxed{\text{RTN}}$	019– 25 12

Slide the PRGM-RUN switch to ▮▮ RUN and evaluate the expression at the following values of x: 1, 2, 7.1935, ln 17.5.

Keystrokes	Display	
1 ENTER↑ ENTER↑		
ENTER↑	1.0000	Fill stack with variable.
A	6.0000	$f(x)$
2 ENTER↑ ENTER↑		
ENTER↑	2.0000	Fill stack with variable.
A	43.0000	$f(x)$
7.1935 ENTER↑		
ENTER↑ ENTER↑	7.1935	Fill stack with variable.
A	3,770.4359	$f(x)$
17.5 f LN ENTER↑		
ENTER↑ ENTER↑	2.8622	Fill stack with variable.
A	139.7118	$f(x)$

Using Horner's Method

As you can see, the above program was logically and easily written; and it produced the results we wanted. However, using a mathematical technique known as Horner's method, we can write a program that is not only logical, but is also simpler and shorter.

For a polynomial expression $a_n x^n + a_{n-1}x^{n-1} + ,\ldots, a_1 x^1 + a_0$, Horner's method essentially reduces all powers $x^n, x^{n-1},\ldots,x^1$ of the variable to x^1. As a result, the expression is stated as a series of arithmetic operations involving the variable x and the coefficients $a_n, a_{n-1},\ldots, a_1, a_0$. For example, applying Horner's method to the polynomial expression we calculated earlier:

$$x^4 + 3x^3 - x^2 + 4x - 1$$

$$(x^3 + 3x^2 - x + 4)x - 1$$

$$((x^2 + 3x - 1)x + 4)x - 1$$

$$(((x + 3)x - 1)x + 4)x - 1$$

We can now write another program using the same stack fill method we used in the previous program. But this time, because we rewrote $f(x)$ using Horner's method, our program involves just seven arithmetic operations instead of the six arithmetic and three exponential operations needed earlier.

Stack Registers

	001	002	003	004	005
T	x	x	x	x	x
Z	x	x	x	x	x
Y	x	x	x	x	$(x+3)x$
X	x	3	$(x+3)$	$(x+3)x$	1
	$\boxed{h}\boxed{LBL}\boxed{B}$	3	$\boxed{+}$	$\boxed{\times}$	1

	006	007	008
T	x	x	x
Z	x	x	x
Y	x	x	$((x+3)x-1)x$
X	$((x+3)x-1)$	$((x+3)x-1)x$	4
	$\boxed{-}$	$\boxed{\times}$	4

	009	010	011
T	x	x	x
Z	x	x	x
Y	x	x	$(((x+3)x-1)x+4)x$
X	$((x+3)x-1)x+4$	$(((x+3)x-1)x+4)x$	1
	$\boxed{+}$	$\boxed{\times}$	1

	012	013
T	x	x
Z	x	x
Y	x	x
X	$(((x+3)x-1)x+4)x-1$	$(((x+3)x-1)x+4)x-1$
	$\boxed{-}$	$\boxed{h}\boxed{RTN}$

The above program uses only 13 lines of program memory, a savings of 6 lines over the previous program to calculate the same expression. Key in the above program and try the same examples we ran earlier.

Slide the PRGM-RUN switch to PRGM ▥█. If you have not executed any other instructions since the previous evaluations of $f(x)$, you will see ***000–*** in the display. If this is not the case, press $\boxed{\text{GTO}}\boxed{\cdot}$ 000 (more on $\boxed{\text{GTO}}$ later).

Keystrokes	**Display**
$\boxed{\text{h}}\ \boxed{\text{LBL}}\ \boxed{\text{B}}$	***001– 25, 13, 12***
3	***002– 3***
$\boxed{+}$	***003– 51***
$\boxed{\times}$	***004– 61***
1	***005– 1***
$\boxed{-}$	***006– 41***
$\boxed{\times}$	***007– 61***
4	***008– 4***
$\boxed{+}$	***009– 51***
$\boxed{\times}$	***010– 61***
1	***011– 1***
$\boxed{-}$	***012– 41***
$\boxed{\text{h}}\ \boxed{\text{RTN}}$	***013– 25 12***

Slide the PRGM-RUN switch to █▥ RUN and evaluate the expression using the same values of x we used earlier.

Keystrokes	**Display**	
1 $\boxed{\text{ENTER↑}}\ \boxed{\text{ENTER↑}}$		
$\boxed{\text{ENTER↑}}$	***1.0000***	Fill stack with variable.
$\boxed{\text{B}}$	***6.0000***	$f(x)$
2 $\boxed{\text{ENTER↑}}\ \boxed{\text{ENTER↑}}$		
$\boxed{\text{ENTER↑}}$	***2.0000***	Fill stack with variable.
$\boxed{\text{B}}$	***43.0000***	$f(x)$
7.1935 $\boxed{\text{ENTER↑}}$		
$\boxed{\text{ENTER↑}}\ \boxed{\text{ENTER↑}}$	***7.1935***	Fill stack with variable.
$\boxed{\text{B}}$	***3,770.4359***	$f(x)$
17.5 $\boxed{\text{f}}\ \boxed{\text{LN}}\ \boxed{\text{ENTER↑}}$		
$\boxed{\text{ENTER↑}}\ \boxed{\text{ENTER↑}}$	***2.8622***	Fill stack with variable.
$\boxed{\text{B}}$	***139.7118***	$f(x)$

Did you notice something different? In addition to reducing the size of the program, using Horner's method reduced execution time as well.

Further Applications

As you have seen, the stack fill technique provides a simple and useful approach to evaluating an expression containing several occurrences of the same variable. By applying Horner's method to the problem, where possible, we realize greater space savings and speed. Later, when we discuss root-finding and numerical integration, you will see how the *automatic* stack fill designed into the SOLVE and $\int_y^x$ operations enhances the power and convenience of your HP-34C's programming and problem-solving capability.

Problems

Using the stack fill technique and Horner's method, write and execute programs for evaluating the following expressions at x values of 1.5, 3.73, and –4.25.

1. $2x^5 - x^4 + 2x^2 + x + 1$
2. $0.97 \text{ Sin}^3 x + 0.04 \text{ Sin}^2 x - 1.73 \text{ Sin} x - 1$

Answers:

1. 17.1250, 1,283.0102, –3,066.5371.
2. –1.0452, –1.1121, –0.8720.

Section 4

Program Editing

Often you may want to alter, correct, or add to a program that is loaded in the calculator. On your HP-34C keyboard, you will find several editing functions that permit you to easily add or change steps in a loaded program *without* reloading the entire program.

As you may recall, there are nine functions that cannot be recorded in program memory. Seven of these functions are *program editing and manipulation functions,* and can aid you in modifying and correcting your programs.

Nonrecordable Operations

CLEAR [PRGM] is one keyboard operation that cannot be recorded in program memory. When you press [f] CLEAR [PRGM] in PRGM mode, program memory is cleared and the calculator is reset to the top of memory (line 000). Note that [f] CLEAR [PRGM] does not reset a RAD or GRD trig mode to DEG.

[SST] *(single step)* is another nonrecordable operation. When you press [h] [SST], in PRGM mode, the calculator moves to and displays the next line in occupied program memory. No program instructions are executed. When you press [h] [SST] in RUN mode the calculator also moves to and displays the next line of program memory. But when you release the [SST] key, the calculator executes the instruction loaded in that line.

[BST] *(back step)* is a nonrecordable operation used in both PRGM and RUN mode to move to and display the previous line of program memory. In RUN mode the original contents of the display reappear when [BST] is released. No program instructions are executed.

CLEAR [PREFIX] is the nonrecordable operation used after a prefix keystroke ([f] , [g] , or [h]) to cancel that keystroke. CLEAR [PREFIX] also cancels all keystrokes in an incomplete instruction such as [f] [SCI] or [STO] [+]. CLEAR [PREFIX] has no effect on a completed instruction (i.e., [f] [SCI] 5, [STO] [+] 1, etc.).

[GTO] *(go to)* [•] *nnn* is used for going to a specific line number, and is another keyboard operation which cannot be loaded as an instruction. (However, [GTO] followed by a numbered label 0 through 9 can be loaded as a program instruction. More about the use of this instruction later.) Whether the calculator is in PRGM or RUN, when you press [GTO] [•] followed by a three digit line number, the program memory is set to that line number. No instructions are executed. If the calculator is in RUN mode, you can verify that the calculator is set to the specified line by briefly switching to PRGM mode. The [GTO] [•] *nnn* operation is especially useful in PRGM mode because it permits you to jump to any location in occupied program memory for editing or checking purposes.

Note: Attempting to execute a [GTO] [•] *nnn* instruction to any lines of program memory that are unoccupied or that the calculator has not converted from data storage registers is an illegal operation which results in the error signal ***Error 4.***

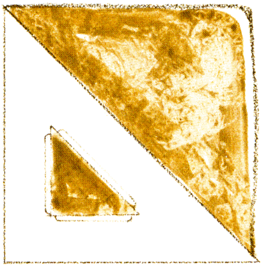
The [DEL] *(delete)* key is a nonrecordable operation that you can use in PRGM mode to delete instructions from program memory. When the calculator is in PRGM mode and you press [h] [DEL], the instruction at the current line of program memory is erased. All subsequent instructions in program memory then move upward one line. Pressing [h] [DEL] in RUN mode does nothing.

The [MEM] *(memory)* function, displays the current memory allocation at any time, in or out of program mode. To review [MEM], see page 59 in section 3, Simple Programming.

Now let's load a program from the keyboard and use your HP-34C's editing tools to check and modify it.

Pythagorean Theorem Program

The following program computes the hypotenuse of any right triangle, given the other two sides. The formula used is $c = \sqrt{a^2 + b^2}$.



Below are instructions for the program (basically, the same keys you would press to solve for c manually), assuming that values for sides a and b have been input to the X- and Y-registers of the stack.

To load the program:

First set the calculator to PRGM mode. Then press f CLEAR PRGM to clear program memory of any previous programs and reset the calculator to line 000 of program memory. Finally, load the program by pressing the keys shown below.

Keystrokes	Display	
h LBL A	*001– 25, 13, 11*	
g x^2	*002– 15 3*	b^2
$x \gtrless y$	*003– 21*	
g x^2	*004– 15 3*	a^2
+	*005– 51*	$a^2 + b^2$
f $\sqrt{x}$	*006– 14 3*	$\sqrt{a^2 + b^2}$
h RTN	*007– 25 12*	End of program; calculator returns to line 000 and halts.

Return the calculator to RUN mode.

Now you can run the program. For example, calculate the hypotenuse of a right triangle with side *a* of 22 meters and side *b* of 9 meters. (Notice that the order of entry does not matter in this case.)

Keystrokes	Display	
22 [ENTER↑]	**22.0000**	
9	**9.**	
[A]	**23.7697**	Length of side *c* in meters.

To compute the hypotenuse of a right triangle with a side *a* of 73 miles and a side *b* of 99 miles:

Keystrokes	Display	
73 [ENTER↑]	**73.0000**	
99	**99.**	
[A]	**123.0041**	Length of side *c* in miles.

Now let's see how we can use the nonrecordable editing features of the calculator to examine and alter this program.

Single-Step Execution of a Program

With the Program Mode switch set to RUN mode, you can execute a recorded program one line at a time by using [h] [SST] *(single-step)*.

To single-step through the Pythagorean Theorem program using a triangle with side *a* of 73 miles and side *b* of 99 miles:

Keystrokes	Display	
73 [ENTER↑]	**73.0000**	
99	**99.**	Program initialized for this set of data before running.

Now, press [h] [SST] and hold [SST] down to see the keycode for the next instruction. When you release the [SST] key, that next instruction is executed. (Remember that the [h] [RTN] instruction in line 007 returned the calculator to line 000 after the last execution of the program.)

Keystrokes	Display	
h SST	*001– 25, 13, 11*	Keycode for h LBL A seen when you hold SST down.
	99.0000	h LBL A executed when you release SST.

(Notice that you didn't have to press A. When you are executing a program one line at a time, pressing h SST begins the program from the current line of program memory; in this case, line 001.)

Continue executing the program by pressing h SST again. When you hold SST down, you see the keycode for the next instruction. When you release SST, that instruction is executed.

Keystrokes	Display	
h SST	*002– 15 3*	Keycode for x^2
	9,801.0000	Executed.

When you press h SST a third time in RUN mode, line 003 of program memory is displayed. When you release the SST key, the instruction in that line, x⇄y, is executed, and the calculator halts.

Keystrokes	Display	
h SST	*003– 21*	Keycode for x⇄y.
	73.0000	Executed.

Continue executing the program by means of h SST. When you have executed the h RTN instruction in line 007, the calculator returns to line 000 (later we will cover more on how RTN works). You have completed executing the program and the answer is displayed, just as if the calculator had executed the program automatically, instead of via h SST.

Keystrokes	Display	
h SST	004– 15 3	
	5329.0000	
h SST	005– 51	
	15,130.0000	
h SST	006– 14 3	
	123.0041	
h SST	007– 25 12	
	123.0041	Final answer.

Note: h SST will not advance into unoccupied lines of program memory. If you single-step from the last *occupied* line of program memory in RUN or PRGM mode your HP-34C conveniently returns to line 000. In RUN mode the original contents of the display remain unchanged. In PRGM mode ***000–*** indicating the top of program memory, is displayed.

You have seen how the SST key can be used in RUN mode to single-step through a program. Using h SST in this manner can help you create and correct programs. Now let's see how you can use GTO • 000, SST, B, and GTO • *nnn* in PRGM mode to help you modify a program.

Modifying a Program

Let's modify the Pythagorean Theorem program so that the X-register contents will automatically be displayed at certain points in the program. We will do this by inserting the h PSE instruction to halt the program and display the contents of the X-register for about 1 second, then resume execution. (More about PSE later.) Here is the program you just ran.

Keystrokes	Display	
h LBL A	001– 25, 13, 11	
g x^2	002– 15 3	
x⇄y	003– 21	We will insert an h PSE instruction after each of these instructions.
g x^2	004– 15 3	
+	005– 51	
f $\sqrt{x}$	006– 14 3	
h RTN	007– 25 12	

Single-Step Viewing Without Execution

You can use [SST] in PRGM mode to single-step to the desired line of program memory without executing the program. When you switch the calculator to PRGM mode, you will see that the calculator is reset to line 000 of program memory as a result of executing the [h] [RTN] instruction in the above example. When you press [h] [SST] once, the calculator now moves to line 001 and displays the contents of that line of program memory. No instructions are executed.

Slide the PRGM-RUN switch to PRGM ▮.

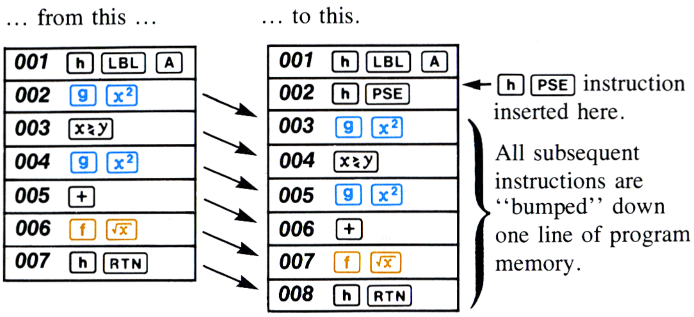
Keystrokes	**Display**	
	000–	Line 000 of program memory.
[h] [SST]	*001– 25, 13, 11*	

You can see that the calculator is set at line 001 of program memory. If you press a *recordable* operation now, it will be loaded in the *next* line, line 002, of program memory, and all subsequent instructions will be "bumped" down one line in program memory.

Thus, to load the [h] [PSE] instruction so that the calculator will pause and display the contents of the X-register:

Keystrokes	**Display**
[h] [PSE]	*002– 25 74*

Now let's see what happened in program memory when you loaded that instruction. With the calculator set at line 001, when you pressed [h] [PSE], program memory was altered ...



Resetting to Line 000

Your HP-34C automatically resets to line 000 when turned on. And, as you know, when you press f CLEAR PRGM with the calculator set to PRGM mode, the calculator is reset to line 000 and all instructions in program memory are erased. However, you can also reset the calculator to line 000 while preserving existing programs in program memory by pressing GTO • 000 in PRGM or RUN mode and h RTN in RUN mode.

To set the calculator to line 000 with the Pythagorean Theorem program loaded into program memory:

Keystrokes	**Display**
GTO • 000	*000-*

Going to a Line Number

It is easy to see that if you wanted to single-step from line 000 to some remote line number in program memory, it would take a great deal of time and a number of presses of h SST. To avoid such inconveniences simply apply the GTO • *nnn* procedure which you used previously to jump to line 000. In a manner similar to GTO • 000, when you press GTO • *nnn*, the calculator immediately jumps to the occupied line number specified by *nnn*. No instructions are executed. If you press GTO • *nnn* in RUN mode, the display remains unchanged. If you press

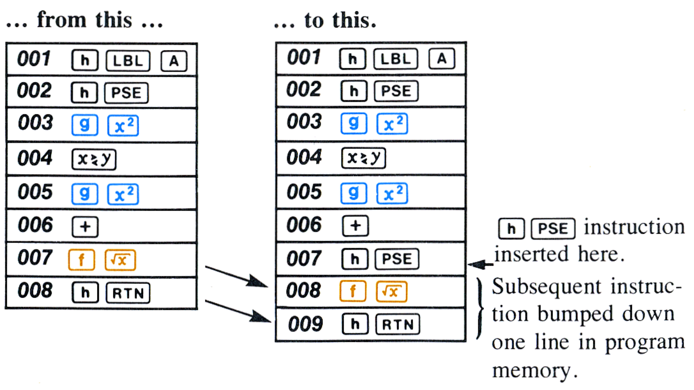
$\boxed{\text{GTO}}$ $\boxed{\cdot}$ *nnn* in PRGM mode, the line of occupied memory (line number and keycodes) addressed by nnn appears in the display. In RUN mode, if you then initiate a label search or program execution, the search or execution will begin with that line of program memory. In PRGM mode, any loading of additional instructions will begin with the next line of program memory.

For example, to add an $\boxed{\text{h}}$ $\boxed{\text{PSE}}$ instruction to review the X-register contents after the squares have been added together by the instruction in line 006, you can first press $\boxed{\text{GTO}}$ *(go to)* followed by a decimal point and the appropriate three-digit line number of program memory. Then press $\boxed{\text{h}}$ $\boxed{\text{PSE}}$ to place that instruction in the *following* line of program memory. Remember that when you add an instruction in this manner, each subsequent instruction is moved down one line in program memory. To add the $\boxed{\text{h}}$ $\boxed{\text{PSE}}$ instruction after the $\boxed{+}$ instruction that is now loaded into line 006, be sure the calculator is in PRGM mode, then:

Keystrokes	**Display**
$\boxed{\text{GTO}}$ $\boxed{\cdot}$ 006	006– 51
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	007– 25 74

As you load the $\boxed{\text{h}}$ $\boxed{\text{PSE}}$ instruction into line 007, the instruction that was *formerly* in line 007 is moved to line 008, and the instruction in subsequent lines are similarly moved down one line.

When you added the $\boxed{\text{h}}$ $\boxed{\text{PSE}}$ instruction after line 006, program memory was altered ...



Inserting Instructions in Longer Programs

After the initial 70 lines of program memory are occupied, the calculator automatically converts storage registers to *available* program memory in blocks of 7 lines at a time. This occurs block by block as each new allocation of program lines is filled up with instructions. If only the first 70 program lines are occupied, inserting another instruction at any point automatically causes the conversion of one storage register ($R_{.9}$ in this case) to 7 more lines of *available* program memory and places the last instruction of the program in line 71. You would then have 77 program lines *available* and 71 *occupied*. (Refer to page 55, Automatic Memory Allocation.) With 77 lines *occupied*, inserting one instruction converts another storage register ($R_{.8}$) to 7 program lines, and so on. If all 210 program lines are *occupied*, the calculator will not accept any additional program instructions. If you attempt to add a new instruction at any point in program memory with all 210 lines already *occupied*, ***Error 4*** appears in the display and program memory remains unchanged. (Remember—pressing [g] [MEM] periodically while loading a long program will tell you the current status of the program line/storage register allocation.)

Stepping Backwards Through a Program

The [BST] *(back step)* key allows you to back step through a loaded program for editing whether the calculator is in RUN or PRGM mode. When you press [h] [BST], the calculator backs up one line in program memory. If the calculator is in RUN mode, the previous line is displayed as long as you hold down the [BST] key. When you release it, the original contents of the X-register are again displayed. In PRGM mode, of course, you can see the line number and keycode of the instruction in the display at all times. No instructions are executed, whether you are in RUN or PRGM mode.

Note: When your HP-34C is at the top of the program memory (line 000), pressing [h] [BST] moves the calculator to the last line of *occupied* program memory. This feature is particularly helpful when you want to quickly verify the length of an existing program or to begin loading a new program or subroutine that you want to follow a program or subroutine already in memory.

You now have one more $\boxed{h}$ $\boxed{PSE}$ instruction to add to the Pythagorean Theorem program. The $\boxed{h}$ $\boxed{PSE}$ instruction should be added after the $\boxed{x \rightleftarrows y}$ instruction that is now loaded in line 004 of program memory. If you have just completed loading $\boxed{h}$ $\boxed{PSE}$ in line 007 as described above, the calculator is set at line 007 of program memory. You can use $\boxed{BST}$ to back the calculator up to line 004, then insert the $\boxed{h}$ $\boxed{PSE}$ instruction in line 005. To begin:

Ensure that the calculator is set to PRGM ▮ mode.

Keystrokes	Display	
	007– 25 74	Calculator initially set to line 008.
$\boxed{h}$ $\boxed{BST}$	*006– 51*	Pressing $\boxed{h}$ $\boxed{BST}$ once moves the calculator back one line in program memory.

Continue using the $\boxed{BST}$ key to move backward through program memory until the calculator displays line 004.

Keystrokes	Display
$\boxed{h}$ $\boxed{BST}$	*005– 15 3*
$\boxed{h}$ $\boxed{BST}$	*004– 21*

Since you wish to insert the $\boxed{h}$ $\boxed{PSE}$ instruction *after* the $\boxed{x \rightleftarrows y}$ instruction now loaded in line 004, you move the calculator to line 004. As always, when you key in an instruction, it is loaded into the next line after the line being displayed. Thus, if you press $\boxed{h}$ $\boxed{PSE}$ now, that instruction will be loaded into line 005 of program memory, and all subsequent instructions will be moved down, or "bumped," one line.

Keystrokes	Display
$\boxed{h}$ $\boxed{PSE}$	*005– 25 74*

You have now finished modifying the Pythagorean Theorem program so that you can review the contents of the X-register at several points while it runs.

The altered program is shown below:

Keystrokes	Display
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$	001 – 25, 13, 11
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	002 – 25 74
$\boxed{\text{g}}$ $\boxed{x^2}$	003 – 15 3
$\boxed{x \gtrless y}$	004 – 21
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	005 – 25 74
$\boxed{\text{g}}$ $\boxed{x^2}$	006 – 15 3
$\boxed{+}$	007 – 51
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	008 – 25 74
$\boxed{\text{f}}$ $\boxed{\sqrt{x}}$	009 – 14 3
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	010 – 25 12

If you wish, you can use $\boxed{\text{h}}$ $\boxed{\text{SST}}$ in PRGM mode to verify that the program in your calculator matches the one shown above.

Running the Modified Program

To run the Pythagorean Theorem program, you need only set the calculator to RUN mode, key in the values for sides *a* and *b* and press $\boxed{\text{A}}$. The calculator displays the X-register contents (side *b*), then squares side *b*, exchanges the contents of the X- and Y-registers, and again reviews the X-register contents (side *a* this time). Then the calculator squares side *a*, adds b^2 to a^2, and reviews the X-register contents ($a^2 + b^2$) a third time. The hypotenuse is then calculated and execution returns to line 000 and halts.

For example, to compute the hypotenuse of a right triangle with sides *a* and *b* of 22 meters and 9 meters:

Set the calculator to RUN .

Keystrokes	Display	
22 $\boxed{\text{ENTER}\uparrow}$	22.0000	
9 $\boxed{\text{A}}$	23.7697	After reviewing the X-register contents three times during the running program, the answer in meters is displayed.

Now run the program for a right triangle with sides *a* and *b* of 73 miles and 99 miles.

(Answer: 123.0041 miles.)

Deleting Instructions

Often in modifying or correcting a program you may wish to delete an instruction from program memory. To delete the instruction to which the calculator is set, merely press the nonrecordable operation $\boxed{\text{h}}\boxed{\text{DEL}}$ *(delete)* with the calculator set to PRGM mode. (When you delete an instruction from program memory using $\boxed{\text{DEL}}$, all subsequent instructions in program memory are moved *up* one line. The calculator then displays the line *preceding* the line that held the instruction you deleted.)

For example, if you wanted to modify the Pythagorean Theorem program that is now loaded into the calculator so that the X-register was only reviewed once, for the sum of the squares, you would have to delete the $\boxed{\text{h}}\boxed{\text{PSE}}$ instructions that are presently loaded in lines 002 and 005 of program memory. To delete these instructions, you must first set the calculator at these lines using $\boxed{\text{h}}\boxed{\text{SST}}$, $\boxed{\text{h}}\boxed{\text{BST}}$, or $\boxed{\text{GTO}}\boxed{\cdot}$ *nnn*, then press $\boxed{\text{h}}\boxed{\text{DEL}}$. To delete the $\boxed{\text{h}}\boxed{\text{PSE}}$ instruction now loaded in line 002:

First, set the calculator to PRGM ▯▮ .

Keystrokes	**Display**	
$\boxed{\text{GTO}}\boxed{\cdot}$ 002	***002– 25 74***	Line 002 is displayed.
$\boxed{\text{h}}\boxed{\text{DEL}}$	***001– 25, 13, 11***	The instruction in line 002 is deleted and the calculator moves to line 001.

You can use $\boxed{\text{h}}\boxed{\text{SST}}$ to verify that the $\boxed{\text{h}}\boxed{\text{PSE}}$ instruction has been deleted and subsequent instructions have been moved up one line.

Keystrokes	**Display**	
$\boxed{\text{h}}\boxed{\text{SST}}$	***002– 15 3***	The instruction formerly in 003 was moved up to line 002, and all subsequent instructions were moved up one line when you pressed $\boxed{\text{h}}\boxed{\text{DEL}}$.

When you set the calculator to line 002 of program memory and pressed $\boxed{\text{h}}\boxed{\text{DEL}}$, memory was altered ...

... from this ...

001	h LBL A
002	h PSE
003	g x^2
004	$x \gtrless y$
005	h PSE
006	g x^2
007	+
008	h PSE
009	f $\sqrt{x}$
010	h RTN

... to this.

001	h LBL A
002	g x^2
003	$x \gtrless y$
004	h PSE
005	g x^2
006	+
007	h PSE
008	f $\sqrt{x}$
009	h RTN

One instruction deleted here.

These instructions all move upward one line.

To delete the h PSE instruction now loaded in line 004 you can use the SST key to single-step down to that line number and then delete the instruction with the h DEL operation.

Keystrokes	**Display**	
h SST	003- 21	
h SST	004- 25 74	
h DEL	003- 21	The h PSE instruction is deleted from line 004 and the calculator displays line 003. Subsequent instructions move up one line of program memory.

If you have modified the program as described above, the X-register is reviewed only once, just after the sum of the squares is calculated. The value of the hypoteneuse is then calculated and execution halts.

Set the calculator to RUN mode and run the program for right triangles with:

Sides *a* and *b* of 17 and 34 meters. After reviewing the X-register (sum of the squares = 1,445.0000 meters), the rest of the program is executed and the calculator halts displaying the hypoteneuse: 38.0132 meters.

Sides *a* and *b* of 550 rods and 740 rods. After reviewing the X-register (sum of the squares = 850,100.0000 rods), the rest of the program is executed and the calculator halts, displaying the hypotenuse: 922.0087 rods.

To replace any instruction with another, simply set the calculator to the desired line of program memory, press [h] [DEL] to delete the first instruction, then press the keystrokes for the new instruction.

When deleting instructions from a program of more than 70 lines, the process of automatically allocating storage registers to program lines works in reverse. For example, deleting any instruction from a 78-line program automatically converts program lines 78-85 back to storage register $R_{.8}$ (Refer to Automatic Memory Allocation, page 55.)

The editing features of the calculator have been designed to provide you with quick and easy access to any part of your program, whether for editing, debugging, or documentation. If a program stops running because of an error or because of an overflow, you can simply switch the calculator to PRGM mode to see the line number and keycode of the operation that caused the error or overflow. If you suspect a portion of your program is faulty, you can use the [GTO] [•] *nnn* operation from the keyboard to go to the suspect section, then use the [SST] operation in RUN mode to monitor every change in calculator status as you execute the program one line at a time.

Problems

1. You may have noticed that there is a single keyboard operation, [g] [→P] , that calculates the hypotenuse, side *c*, of a right triangle with sides *a* and *b* input to the X- and Y-registers. Replace the [x^2], [x⇄y], [x^2], [+], [PSE], and [$\sqrt{x}$] instructions in the Pythagorean Theorem program with the single [g] [→P] instruction as follows:

a. Use $\boxed{\text{GTO}}$ $\boxed{\bullet}$ *nnn* and $\boxed{\text{h}}$ $\boxed{\text{SST}}$ to verify that the Pythagorean Theorem program contains the instructions shown below.

Keystrokes	**Display**
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$	001– 25, 13, 11
$\boxed{\text{g}}$ $\boxed{x^2}$	002– 15 3
$\boxed{x \gtrless y}$	003– 21
$\boxed{\text{g}}$ $\boxed{x^2}$	004– 15 3
$\boxed{+}$	005– 51
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	006– 25 74
$\boxed{\text{f}}$ $\boxed{\sqrt{x}}$	007– 14 3
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	008– 25 12

Replace all of these instructions with a $\boxed{\text{g}}$ $\boxed{\to\text{P}}$ instruction.

b. Use the $\boxed{\text{GTO}}$ $\boxed{\bullet}$ *nnn* keyboard operation to go to line 007, the last instruction to be deleted in the program.

c. Use the $\boxed{\text{h}}$ $\boxed{\text{DEL}}$ keyboard operation in PRGM mode to delete the instruction in lines 007, 006, 005, 004, 003, and 002.

d. Load the $\boxed{\text{g}}$ $\boxed{\to\text{P}}$ instruction into line 002.

e. Verify that the modified program looks like the one below.

$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$	001– 25, 13, 11
$\boxed{\text{g}}$ $\boxed{\to\text{P}}$	002– 15 4
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	003– 25 12

f. Switch to RUN mode and run the program for a right triangle with sides *a* and *b* of 73 feet and 112 feet. (Answer: 133.6899 feet.)

2. The following program is used by the manager of a savings and loan company to compute the future amounts of savings accounts according to the formula $FV = PV(1 + i)^n$, where *FV* is future value or amount, *PV* is present value, *i* is the periodic interest rate expressed as a decimal, and *n* is the number of periods. With *PV* entered into the Y-register, *n* keyed into the X-register, and an annual interest rate of 7.5%, the program is:

Keystrokes	Display
h LBL B	001 – 25, 13, 12
f FIX 2	002 – 14, 11, 2
1	003 – 1
•	004 – 73
0	005 – 0
7	006 – 7
5	007 – 5
x⇄y	008 – 21
h y^x	009 – 25 3
×	010 – 61
h RTN	011 – 25 12

a. Load the program into the calculator.

b. Run the program to find the future amount of $1,000 invested for 5 years.

(Answer: $1,435.63)

Of $2,300 invested for 4 years.

(Answer: $3,071.58)

c. Alter the program to account for a change of the annual interest rate from 7.5% to 8%.

d. Run the program for the new interest rate to find the future value of $500 invested for 4 years; of $2,000 invested for 10 years.

(Answer: $680.24; $4,317.85)

3. The following program calculates the time it takes for an object to fall to the earth when dropped from a given height. (Friction from the air is not taken into account.) When the height *h* in meters is keyed into the displayed X-register and B is pressed, the time *t* in seconds the object to fall to earth is computed according to the formula:

$$t = \sqrt{\frac{2h}{9.8}}$$

a. Clear all previously recorded programs from the calculator, reset the display mode to Fix 4, and load the program below.

Keystrokes	Display
f CLEAR PRGM	000–
h LBL B	001– 25, 13, 12
2	002– 2
×	003– 61
9	004– 9
•	005– 73
8	006– 8
÷	007– 71
f √x	008– 14 3
h RTN	009– 25 12

b. Run the program to compute the time taken by a stone to fall from the top of the Eiffel Tower, 300.51 meters high; and from a blimp stationed 1000 meters in the air.

(Answers: 7.8313 seconds; 14.2857 seconds.)

c. Alter the program to compute the time of descent when the height in *feet* is known, according to the formula:

$$t = \sqrt{\frac{2h}{32.1740}}$$

d. Run the altered program to compute the time taken by a stone to fall from the top of the Grand Coulee Dam, 550 feet high; and from the 1350-foot height of the World Trade Center buildings in New York City.

(Answers: 5.8471 seconds; 9.1607 seconds.)

Branching, Decisions, and Flags

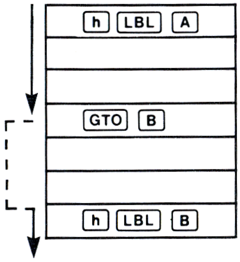
Unconditional Branching and Looping

You have seen how the nonloadable operation $\boxed{\text{GTO}}$ $\boxed{\bullet}$ *nnn* can be used from the keyboard to transfer to any line in occupied program memory. You can also use the *go to* instruction as part of a program. However, in order for $\boxed{\text{GTO}}$ to be *recorded* as an instruction, it must be followed by a label designator ($\boxed{\text{A}}$ or $\boxed{\text{B}}$, or 0 through 9). (It can also be followed by $\boxed{\text{I}}$—more about using $\boxed{\text{I}}$ later.)

When the calculator is executing a program and encounters a $\boxed{\text{GTO}}$ $\boxed{\text{B}}$ instruction, for example, it immediately halts execution and begins searching sequentially downward through program memory for that label. When the first $\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{B}}$ instruction is then encountered, execution resumes.

By using a $\boxed{\text{GTO}}$ instruction followed by a label designator in a program, you can transfer execution to any part of the program that you choose.

Execution branches to next $\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{B}}$.



A $\boxed{\text{GTO}}$ instruction used this way is known as an *unconditional branch*. It always *branches* execution from the $\boxed{\text{GTO}}$ instruction to the specified label. (Later, you will see how a conditional instruction can be used in conjunction with a $\boxed{\text{GTO}}$ instruction to create a *conditional* branch—a branch that depends on the outcome of a test.)

A common use of a branch is to create a "loop" in a program. For example, the following program calculates and displays the square roots of consecutive whole numbers beginning with the number 1. The HP-34C continues to compute the square root of the next consecutive whole number until you press R/S to stop program execution (or until the calculator overflows).

To key in the program:

First, slide the PRGM-RUN switch to PRGM. Press f CLEAR PRGM to clear program memory and to reset the calculator to line 000.

Keystrokes	Display	
h LBL A	*001– 25, 13, 11*	
0	*002– 0*	
STO 1	*003– 23 1*	
h LBL 0	*004– 25, 13, 0*	
1	*005– 1*	
STO + 1	*006– 23, 51, 1*	Adds 1 to current number in R_1.
RCL 1	*007– 24 1*	Recalls current number from R_1.
h PSE	*008– 25 74*	Displays current number.
f √x	*009– 14 3*	
h PSE	*010– 25 74*	Displays square root of current number.
GTO 0	*011– 22 0*	Transfers execution to h LBL 0.
h RTN	*012– 25 12*	

To run the program, slide the PRGM-RUN switch to RUN and press A. The program will begin displaying a table of integers and their square roots and will continue until you press R/S from the keyboard or until the calculator overflows.

How it works: When you press A, the calculator searches through program memory until it encounters the h LBL A instruction that begins the program. It executes that instruction and each subsequent instruction in order until it reaches line 011, the GTO 0 instruction.

The $\boxed{\text{GTO}}$ 0 instruction causes the calculator to *search* once again, this time for a $\boxed{\text{LBL}}$ 0 instruction in the program. When it encounters the $\boxed{\text{LBL}}$ 0 instruction loaded in line 004, execution begins again from $\boxed{\text{LBL}}$ 0. (Notice that the address after a $\boxed{\text{GTO}}$ instruction in a program is a *label*, not a line number.) Since execution is transferred to the $\boxed{\text{LBL}}$ 0 instruction in line 004 each time the calculator executes the $\boxed{\text{GTO}}$ 0 instruction in step 011, the calculator will remain in this "loop," continually adding one to the number in storage register R_1 and displaying the new number and its square root.

Looping techniques like the one illustrated here are common and extraordinarily useful in programming. By using loops, you take advantage of one of the most powerful features of the HP-34C—the ability to update data and perform calculations automatically, quickly, and, if you so desire, endlessly.

You can use unconditional branches to create a loop, as shown above, or in any part of a program where you wish to transfer execution to another label. When the calculator executes a $\boxed{\text{GTO}}$ instruction in a running program it searches sequentially downward through program memory and begins execution again at the first specified label it encounters.

In RUN mode you can also use $\boxed{\text{GTO}}$ for finding a label without running the program in memory. When you execute $\boxed{\text{GTO}}$ ($\boxed{\text{A}}$, $\boxed{\text{B}}$, or *n*) from the keyboard you cause the calculator to go to the specified label and halt. This feature is convenient when you want to simply review or edit lines of memory following a certain label instead of executing them as part of a program.

Problems

1. The following program calculates and pauses to display the square of the number in storage register R_1 each time it is run. Key the program in with the PRGM-RUN switch set to PRGM ▥■, then switch to RUN mode and run the program a few times to see how it works. (The answer will always be 1.000.) Finally, modify the program by inserting a $\boxed{\text{GTO}}$ 1 instruction after the $\boxed{\text{h}}$ $\boxed{\text{PSE}}$ instruction at line 009 and inserting an $\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 1 instruction after the $\boxed{\text{STO}}$ 1 instruction in line 003. This will create a loop that will continually display a new number and its square, then increment

the number by 1, display the new number and compute and display its square, etc. To load the original program, before modification, slide the PRGM-RUN switch to PRGM. Then press:

Keystrokes	Display
f CLEAR PRGM	000–
h LBL B	001– 25, 13, 12
0	002– 0
STO 1	003– 23 1
1	004– 1
STO + 1	005– 23, 51, 1
RCL 1	006– 24 1
h PSE	007– 25 74
g x^2	008– 15 3
h PSE	009– 25 74
h RTN	010– 25 12

Slide the PRGM-RUN switch to RUN and test the program in its original form. After keying in the suggested modifications, run the program again to generate a table of squares.

2. Use the following flowchart to create a program that computes and pauses to display the future value (*FV*) of a compound interest savings account in increments of one year according to the formula:

$$FV = PV(1 + i)^n$$

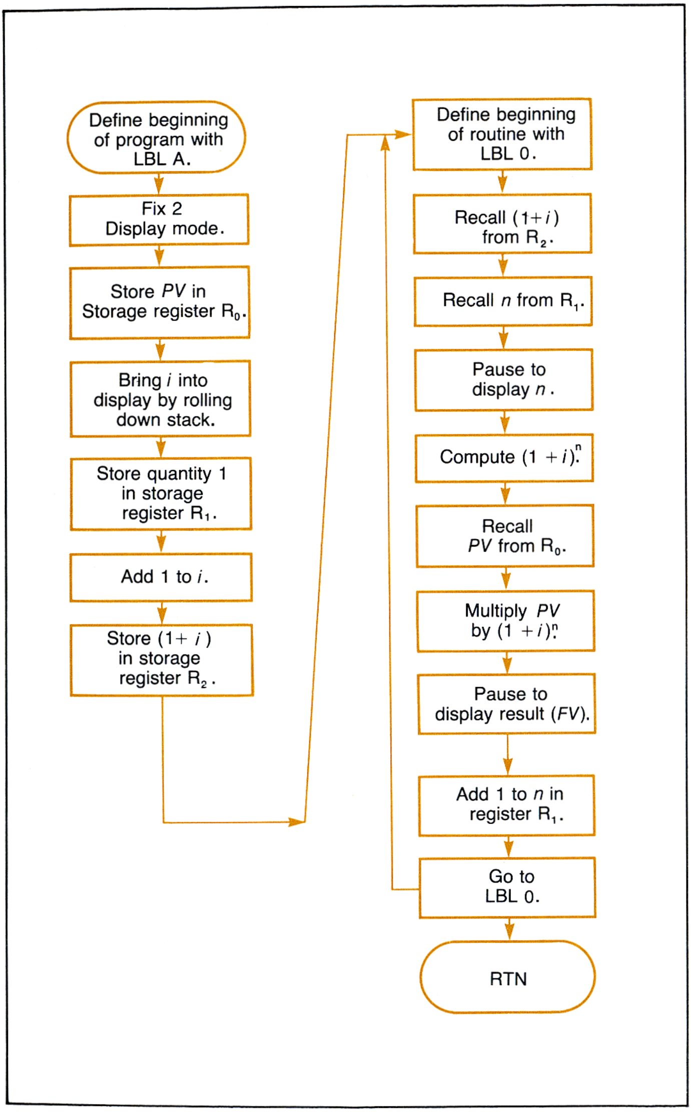
where FV = future value of the savings account.
PV = present value (or principal) of the account.
i = interest rate (expressed as a decimal fraction; e.g., 6% is expressed as 0.06).
n = number of compounding periods (usually, years).

Assume that program execution will begin with *i* entered into the Y-register of the stack and with *PV* keyed into the displayed X-register.

After you have written and loaded the program, run it for an initial interest rate *i* of 6% (keyed in as .06) and an initial deposit (or present value, *PV*) of $1000.

(Answer: 1st year, $1060; 2nd year, $1123.60; 3rd year, $1191.02; etc.)

The program will continue running until you press **R/S** (or any key), or until the HP-34C overflows. You can see how your savings would grow from year to year. Try the program for different interest rates *i* and values of *PV*.


Define beginning of program with LBL A.
Fix 2 Display mode.
Store *PV* in Storage register R_0.
Bring *i* into display by rolling down stack.
Store quantity 1 in storage register R_1.
Add 1 to *i*.
Store (1+ *i*) in storage register R_2 .
Define beginning of routine with LBL 0.
Recall (1+ *i*) from R_2.
Recall *n* from R_1.
Pause to display *n* .
Compute $(1 + i)^n$.
Recall *PV* from R_0.
Multiply *PV* by $(1 + i)^n$.
Pause to display result (*FV*).
Add 1 to *n* in register R_1.
Go to LBL 0.
RTN

Solutions:

Here is a possible solution to problem 2.

h LBL A	
f FIX 2	Convenient financial display mode.
STO 0	Stores *PV*.
g R↓	Brings *i* into display.
1	
STO 1	Stores initial quantity (1) of *n*.
+	Adds 1 to *i*.
STO 2	Stores (1 + *i*).
h LBL 0	
RCL 2	Recalls (1 + *i*).
RCL 1	Recalls *n*.
h PSE	Displays *n*.
h y^x	Calculates $(1 + i)^n$.
RCL 0	Recalls original *PV* from R_0.
×	Calculates new *FV*.
h PSE	Displays result (*FV*).
1	
STO + 1	Adds 1 to *n* in R_1.
GTO 0	Unconditional branch.
h RTN	End of Program.

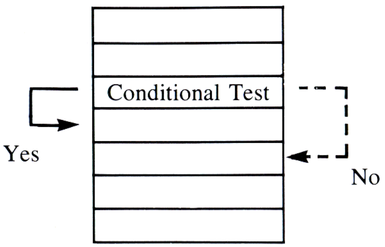
Conditionals and Conditional Branches

Often there are times when you want a program to make a decision. The *conditional* operations on your HP-34C keyboard are program instructions that allow your calculator to make decisions. The conditionals available on your HP-34C are:

f x≤y	tests to see if the value in the X-register is less than or equal to the value in the Y-register.

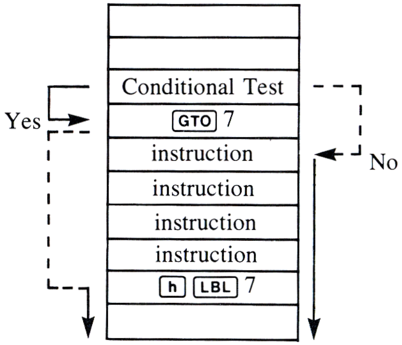
f x>y	tests to see if the value in the X-register is greater than the value in the Y-register.
f x≠y	tests to see if the value in the X-register is not equal to the value in the Y-register.
f x=y	tests to see if the value in the X-register is equal to the value in the Y-register.
g x<0	tests to see if the value in the X-register is less than zero.
g x>0	tests to see if the value in the X-register is greater than zero.
g x≠0	tests to see if the value in the X-register is not equal to zero.
g x=0	tests to see if the value in the X-register is equal to zero.
h F? *n*	tests to see if flag *n* is set (more on flags later).

Each conditional essentially asks a question when it is encountered as an instruction in a program. If the answer is YES, program execution continues sequentially downward with the next line in program memory. If the answer is NO, the calculator branches *around* the next line. For example:



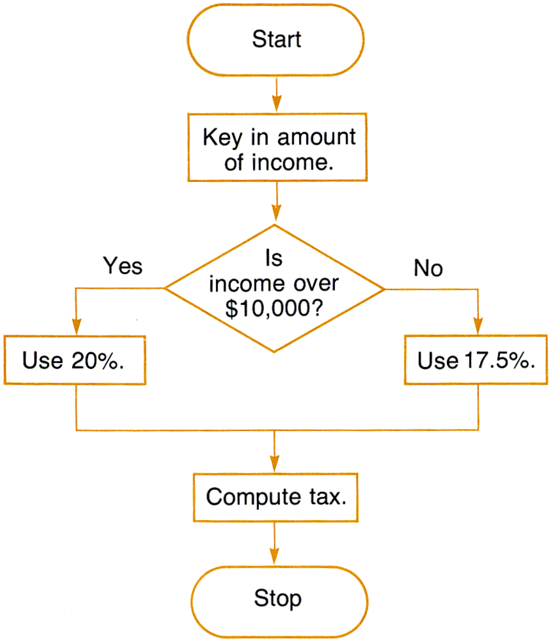
You can see that after it has made the conditional test, the calculator will do the next instruction if the test is true. This is the "DO IF TRUE" rule.

The line immediately following the conditional test can contain any instruction. The most commonly used instruction you'll find will be a GTO instruction. This will branch program execution to another section of program memory if the conditional test is true.



Example: Certified Public Accountant Polly Preparer knows that persons with incomes over $10,000 pay a tax of 20% and persons with incomes of $10,000 or less pay a tax of 17.5%. To make her job easier, Preparer wants to write a program that will allow her to compute taxes for all her clients in the simplest way possible. She will use a program containing conditional branches.

The flowchart for the program might look like this:



To key in the program:

Slide the PRGM-RUN switch to PRGM ▮.

Keystrokes	Display	
$\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$	*000–*	
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$	*001– 25, 13, 11*	
$\boxed{\text{EEX}}$	*002– 33*	Amount of $10,000 placed in Y-register.
4	*003– 4*	
$\boxed{x \gtrless y}$	*004– 21*	
$\boxed{\text{f}}$ $\boxed{x>y}$	*005– 14 51*	If amount of income is greater than $10,000, go to portion of program defined by label B.
$\boxed{\text{GTO}}$ $\boxed{\text{B}}$	*006– 22 12*	
1	*007– 1*	Tax percentage for this portion of program is 17.5.
7	*008– 7*	
$\boxed{\cdot}$	*009– 73*	
5	*010– 5*	
$\boxed{\text{GTO}}$ 1	*011– 22 1*	
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{B}}$	*012– 25, 13, 12*	
2	*013– 2*	Tax percentage for this portion of program is 20.
0	*014– 0*	
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 1	*015– 25, 13, 1*	
$\boxed{\text{h}}$ $\boxed{\%}$	*016– 25 41*	
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	*017– 25 12*	

To run the program to compute taxes on incomes of $15,000 and $7,500:

Slide the PRGM-RUN switch to ▮ RUN .

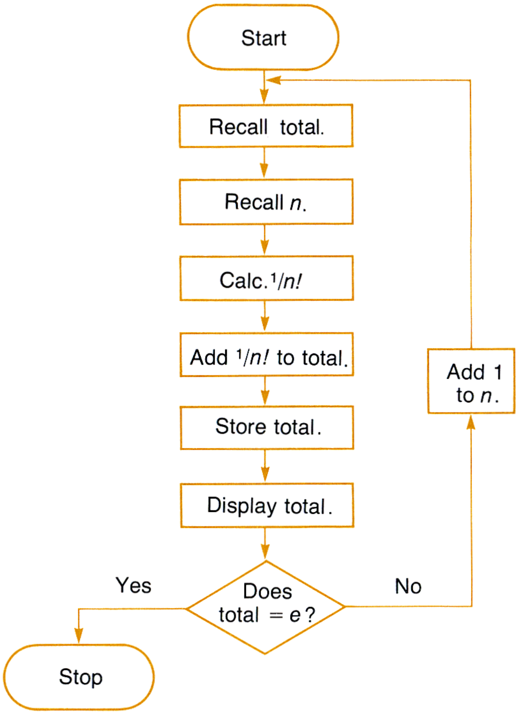
Keystrokes	**Display**	
15000 $\boxed{\text{A}}$	***3,000.00***	Dollars of tax.
7500 $\boxed{\text{A}}$	***1,312.50***	Dollars of tax.

All Preparer has to do to compute tax rates for her other clients is key in their incomes and press **A**. The calculator automatically determines the clients' tax bracket and computes the tax.

Another place where you often want a program to make a decision is within a loop. The loops that you have seen have, to this point, been *infinite* loops—that is, once the calculator begins executing a loop, it remains locked in that loop, executing the same set of instructions over and over again, forever (or, more practically, until the calculator overflows or you halt the running program by pressing **R/S** or any other key).

You can use the decision-making power of the conditional instructions to shift program execution out of a loop. A conditional instruction can shift execution out of a loop after a specified number of iterations or when a certain value has been reached within the loop.

Example: As you know, your HP-34C contains a value for *e*, the base of natural logarithms. (You can display the calculator's value for *e* by pressing 1 **g** **e^x**.) The following program shows that $1/n!$ can be used to verify that the series $e = 1/0! + 1/1! + 1/2! + \ldots + 1/n!$ approximates the value for *e*. After each iteration through the loop, the latest approximation is displayed and compared to the *calculator's* value for *e*. When the two values are equal, the execution is transferred out of the loop to stop the program.


Start
Recall total.
Recall n.
Calc. 1/n!
Add 1/n! to total.
Store total.
Display total.
Does total = e ?
Yes
No
Add 1 to n.
Stop

To load the program into the calculator:

Slide the PRGM-RUN switch to PRGM ▯█ .

Keystrokes	Display
$\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$	*000–*
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$	*001– 25, 13, 11*
$\boxed{\text{RCL}}$ 1	*002– 24 1*
$\boxed{\text{RCL}}$ 0	*003– 24 0*
$\boxed{\text{h}}$ $\boxed{x!}$	*004– 25 1*
$\boxed{\text{h}}$ $\boxed{1/x}$	*005– 25 2*
$\boxed{+}$	*006– 51*
$\boxed{\text{f}}$ $\boxed{\text{FIX}}$ 9	*007– 14, 11, 9*
$\boxed{\text{STO}}$ 1	*008– 23 1*
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	*009– 25 74*
1	*010– 1*
$\boxed{\text{g}}$ $\boxed{e^x}$	*011– 15 1*
$\boxed{\text{f}}$ $\boxed{x{=}y}$	*012– 14 71*
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	*013– 25 12*
1	*014– 1*
$\boxed{\text{STO}}$ $\boxed{+}$ 0	*015– 23, 51, 0*
$\boxed{\text{GTO}}$ $\boxed{\text{A}}$	*016– 22 11*

Slide the PRGM-RUN switch to █▯ RUN .

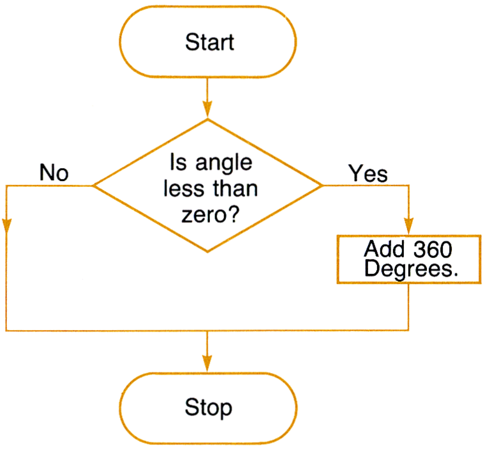
Ensure that the registers are cleared to zero. Then press $\boxed{\text{A}}$ to run the program.

Keystrokes	Display	
$\boxed{\text{f}}$ CLEAR $\boxed{\text{REG}}$	*1,312.50*	Clears all storage registers to zero. (Displayed value remains from previous example.)
$\boxed{\text{A}}$	*2.718281828*	

You can see that execution continues within the loop until the approximation for *e* equals the calculator's value for *e*. When the instruction $\boxed{x{=}y}$ in line 012 is finally true, execution is transferred out of the loop.

Problems

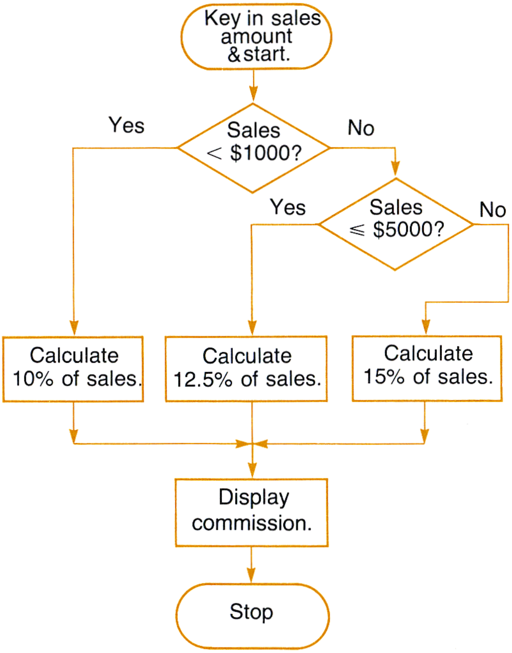
1. Write a program that tests for a negative angle and then converts any negative angle to its positive equivalent. Use a conditional, and, if the angle is negative, add 360 degrees to it to make the angle positive. Use the flowchart below to help you write the program.



2. Use the flowchart to help you write a program that will allow a dealer to compute sales staff commissions at the rates of 10% of sales of up to $1000, 12.5% for sales of $1000 to $5000, and 15% for sales of over $5000. The program should display the amount of sales and the amount of commission.

Load the program and run it for sales amounts of $500, $1000, $1500, $5000, and $6000.

(Answers: $50.00, $125.00, $187.50, $625.00, $900.00.)


Key in sales amount & start.
Sales < $1000?
Yes
No
Sales ≤ $5000?
Yes
No
Calculate 10% of sales.
Calculate 12.5% of sales.
Calculate 15% of sales.
Display commission.
Stop

Solutions:

1.

Keystrokes	Display
f CLEAR PRGM	000-
h LBL A	001- 25, 13, 11
g x<0	002- 15 41
GTO 0	003- 22 0
h RTN	004- 25 12
h LBL 0	005- 25, 13, 0
3	006- 3
6	007- 6
0	008- 0
+	009- 51
h RTN	010- 25 12

User instructions: After keying in program, set PRGM-RUN switch to RUN and set display mode to FIX 4. Input angle and press A.

2.

Keystrokes	Display
f CLEAR PRGM	000-
h LBL A	001- 25, 13, 11
EEX	002- 33
3	003- 3
f x>y	004- 14 51
GTO 0	005- 22 0
5	006- 5
×	007- 61
x⇄y	008- 21
f x≤y	009- 14 41
GTO 1	010- 22 1
1	011- 1
5	012- 5
h %	013- 25 41
h RTN	014- 25 12
h LBL 1	015- 25, 13, 1

Keystrokes	Display
1	016- 1
2	017- 2
[•]	018- 73
5	019- 5
[h] [%]	020- 25 41
[h] [RTN]	021- 25 12
[h] [LBL] 0	022- 25, 13, 0
[x≷y]	023- 21
1	024- 1
0	025- 0
[h] [%]	026- 25 41
[h] [RTN]	027- 25 12

User instructions: After keying in program, set PRGM-RUN switch to ▮▮▮ RUN and set display mode to FIX 2. Input sales dollars and press [A].

Flags

Besides the conditionals ([x=y], [x>0], etc.), you can also use flags for tests in your programs. A flag actually is a memory device that can be either SET (true) or CLEAR (false). A running program can then test the flag later in the program and make a decision, depending upon whether the flag was set or clear.

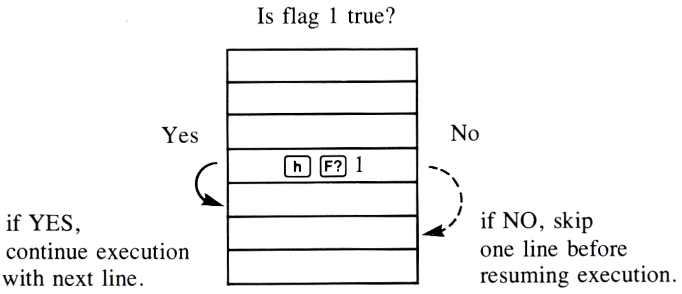
There are four flags available in your HP-34C. They are numbered 0, 1, 2, and 3. To set a flag true, use the instruction [SF] *(set flag)* followed by the proper digit key (0, 1, 2, or 3) of the desired flag. To set *flag 3*, for example, you would use these keystrokes:

[h] [SF] 3

Flags are cleared using the [CF] *(clear flag)* instruction followed by the proper digit key. To clear flag 3 you would use these keystrokes:

[h] [CF] 3

When using flags, decisions are made using the instruction [F?] *(is flag true?)* followed by the digit key (0, 1, 2, 3) specifying the flag to be tested. When a flag is tested by a [h] [F?] *n* instruction, the calculator executes the next line if the flag is set (this is the "DO if TRUE" rule). If the flag is clear, the next line of program memory is skipped before execution resumes.


Is flag 1 true?
Yes
No
h F? 1
if YES, continue execution with next line.
if NO, skip one line before resuming execution.

A flag which has been set by an [h] [SF] *n* command remains set until it is cleared by one of the following:*

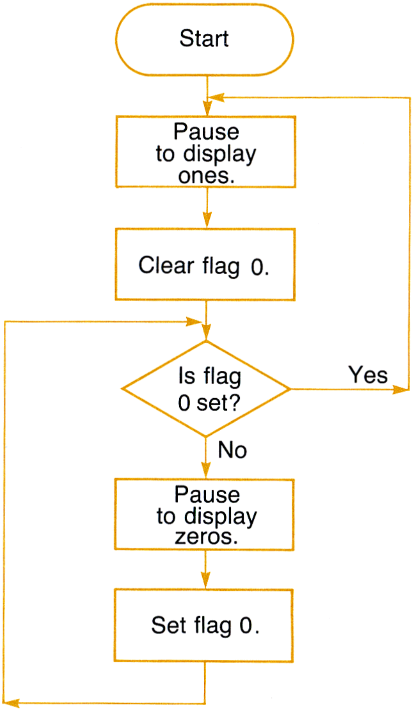
1. Executing an [h] [CF] *n* command.
2. Turning the calculator OFF.

Using flags

Like the *the x/y* and *x/0* conditional tests, flags give you the capability to either skip or execute individual lines in program memory. However, while the *x/y* and *x/0* conditionals function by comparing values, flags function by telling the calculator whether or not a particular operation or type of operation has been performed.

* Note that pressing [f] CLEAR [PRGM] does not clear a flag that has been set by an [h] [SF] *n* instruction.

Example: The following program contains an infinite loop that illustrates the operation of a flag. The program alternately displays all 1's and all 0's by changing the status of the flag, and hence, the result of the test in line 006, each time through the loop. A flowchart for the program might look like this:



The program assumes that you have stored the number 0 in register R_0 and the number 1.111111111 in register R_1.

Slide the PRGM-RUN switch to PRGM .

Keystrokes	Display	
$\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$	*000–*	
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$	*001– 25, 13, 11*	
$\boxed{\text{RCL}}$ 1	*002– 24 1*	Recalls and displays ones from register R_1.
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	*003– 25 74*	
$\boxed{\text{h}}$ $\boxed{\text{CF}}$ 0	*004– 25, 61, 0*	Clears flag 0.
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{B}}$	*005– 25, 13, 12*	
$\boxed{\text{h}}$ $\boxed{\text{F?}}$ 0	*006– 25, 71, 0*	Tests flag 0.
$\boxed{\text{GTO}}$ $\boxed{\text{A}}$	*007– 22 11*	If set (true), go to $\boxed{\text{LBL}}$ $\boxed{\text{A}}$.
$\boxed{\text{RCL}}$ 0	*008– 24 0*	Otherwise, recall and display zeros from register R_0,
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	*009– 25 74*	
$\boxed{\text{h}}$ $\boxed{\text{SF}}$ 0	*010– 25, 51, 0*	set flag 0, and go to
$\boxed{\text{GTO}}$ $\boxed{\text{B}}$	*011– 22 12*	$\boxed{\text{LBL}}$ $\boxed{\text{B}}$.
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	*012– 25 12*	

Now switch to RUN , load storage registers R_0 and R_1, then execute the program.

Keystrokes	Display	
$\boxed{\text{f}}$ $\boxed{\text{FIX}}$ 9	*0.000000000*	
0 $\boxed{\text{STO}}$ 0	*0.000000000*	
1.111111111	*1.111111111*	
$\boxed{\text{STO}}$ 1	*1.111111111*	
$\boxed{\text{A}}$	*1.111111111*	All ones and all zeros.
	0.000000000	

To stop the running program, press $\boxed{\text{R/S}}$ (or any other key).

How it works: After you have initialized the program by storing zero in register R_0 and all ones in register R_1, the program begins running when you press [A]. The [RCL] 1 and [h] [PSE] instructions in lines 002 and 003 pause to display all ones from storage register R_1. The [h] [CF] 0 instruction in line 004 clears flag 0. (Since the flag is already clear when you begin the program, the status of the flag simply remains the same.)

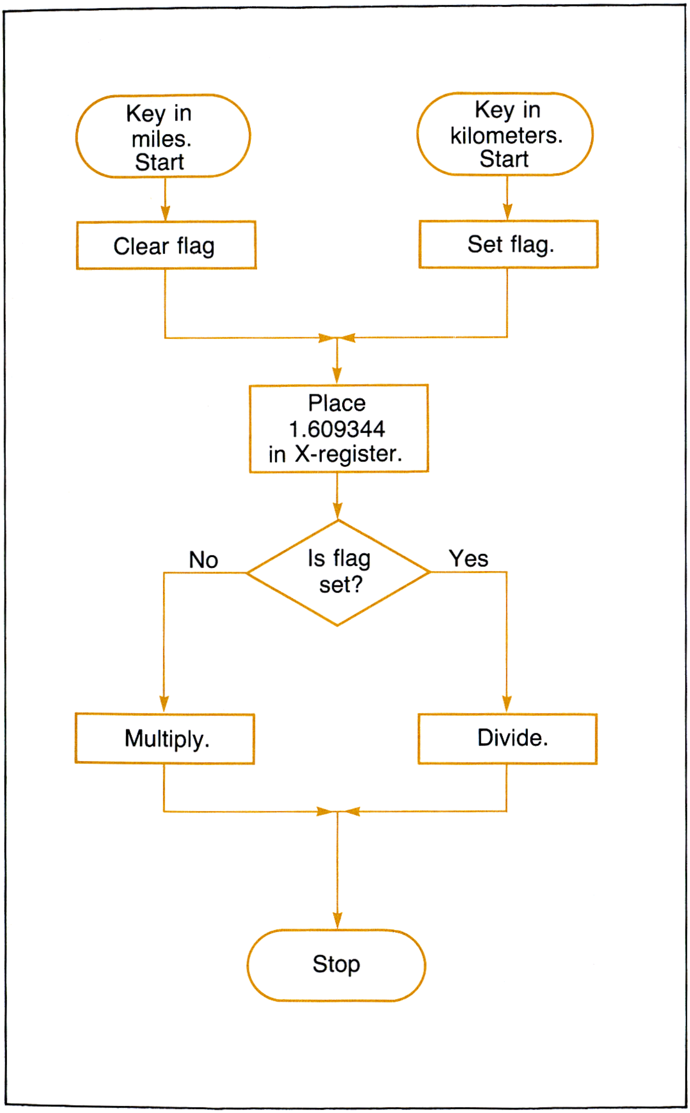
There is no [RTN] after the routine begun by [LBL] [A], so execution continues through the [LBL] [B] instruction in line 005 to the test, [h] [F?] 0, in line 006. The [h] [F?] 0 instruction asks the question "Is flag 0 set (true)?" Since the flag has been cleared earlier, the answer is NO, and execution skips one line of program memory and continues with the [RCL] 0 instruction in line 008. The [RCL] 0 and [h] [PSE] instructions in lines 008 and 009 pause to display all zeros from register R_0. Flag 0 is then set by the [h] [SF] 0 instruction in line 010 and execution is transferred to [LBL] [B] by the [GTO] [B] instruction in line 011.

With flag 0 now set, the answer to the test [h] [F?] 0 ("Is flag 0 true?") is now YES, so the calculator executes the [GTO] [A] instruction in line 007, the next line after the test. After again pausing to display all ones, the flag is cleared, and the program continues in an endless cycle, alternately displaying ones and zeros, until you stop execution from the keyboard.

Problem

One mile is equal to 1.609344 kilometers. Use the following flowchart to create and load a program that will permit you to key in distance in either miles (define with [LBL] [A]) or kilometers (define with [LBL] [B]). Use a flag for determining whether to multiply or divide to convert from one unit of measure to the other. (Hint: [h] [1/x] [×] yields the same result as [÷].)

Set the calculator to [FIX] 4 display mode. Then run the program to convert 26 miles into kilometers; to convert 1500 meters (1.5 kilometers) into miles. (Answers: 41.8429 kilometers; 0.9321 miles.)


Key in miles. Start
Key in kilometers. Start
Clear flag
Set flag.
Place 1.609344 in X-register.
No
Is flag set?
Yes
Multiply.
Divide.
Stop

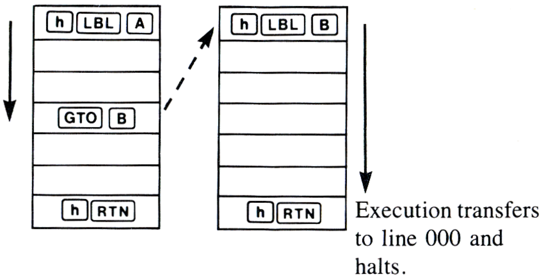
Section 6

Subroutines

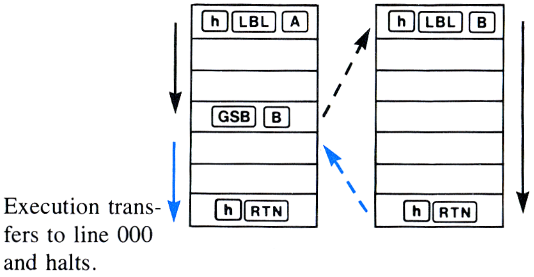
Often, a program contains a certain series of instructions that are executed several times throughout the program. When the same set of instructions occurs more than once in a program, it can be executed as a subroutine. A subroutine is selected by the $\boxed{\text{GSB}}$ *(go to subroutine)* operation, followed by a label address $\boxed{\text{A}}$, $\boxed{\text{B}}$, or 0 through 9. You can also select a subroutine with $\boxed{\text{GSB}}$ $\boxed{\text{I}}$—more about $\boxed{\text{I}}$ later.

A $\boxed{\text{GSB}}$ instruction transfers execution to the routine specified by the label address, just like a $\boxed{\text{GTO}}$ instruction. However, after a $\boxed{\text{GSB}}$ instruction has been executed, when the running program then executes a $\boxed{\text{RTN}}$ *(return)*, execution is transferred back to the next instruction after the $\boxed{\text{GSB}}$. Execution then continues sequentially downward through program memory. The illustration below should make the distinction between $\boxed{\text{GTO}}$ and $\boxed{\text{GSB}}$ more clear.

Branch



Subroutine

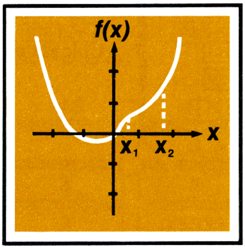


In the top illustration of a branch, if you pressed [A] from the keyboard, the program would execute instructions sequentially downward through program memory. If it encountered a [GTO] [B] instruction, it would then search for the next [LBL] [B] and continue execution from there, until it encountered a [RTN]. When it executed the [RTN] instruction, execution would transfer directly to line 000 and halt.

However, if the running program encounters a [GSB] [B] *(go to subroutine B)* instruction, as shown in the lower illustration, it searches downward for the next [LBL] [B] and resumes execution. When it encounters a [RTN] *(return)*, program execution is once again transferred, this time back to the first line after the origin of the subroutine call ([GSB] [B]), where execution resumes.

As you can see, the only difference between a subroutine and a normal branch is the transfer of execution *after* the [RTN]. After a [GTO], the next [RTN] causes execution to transfer to line 000 and halt; after a [GSB], the next [RTN] returns execution back to the main program, where it continues until another [RTN] (or a [R/S]) is encountered.

Example: Write a program for calculating the average slope of the graph of $f(x)$ between x_1 and x_2 where $f(x) = x^2 - \ln(x^2 + e^{-x})$.



Solution: The average slope of $f(x)$ between x_1 and x_2 is given by

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$= \frac{\left[x_2^2 - \ln(x_2^2 + e^{-x_2})\right] - \left[x_1^2 - \ln(x_1^2 + e^{-x_1})\right]}{x_2 - x_1} .$$

Notice that the solution requires two computations of the expression $x^2 - \ln(x^2 + e^{-x})$.

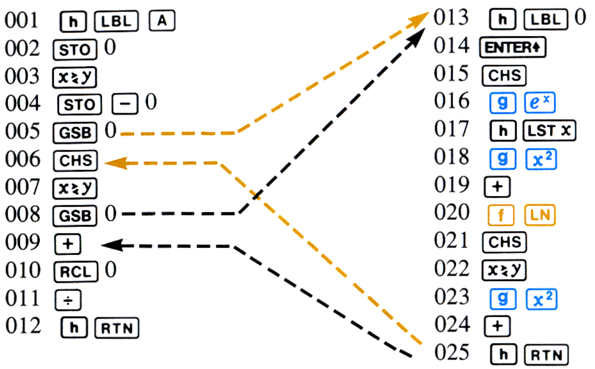
The program below allows you to key in the values for x_1 and x_2 and compute the average slope by pressing [A].

```
[h] [LBL] [A]
[STO] 0
[x⇄y]
[STO] [-] 0
[ENTER↑]   ⎫
[CHS]      ⎪
[g] [eˣ]   ⎪
[h] [LST x]⎪
[g] [x²]   ⎪
[+]        ⎬ Calculates f(x₁) .
[f] [LN]   ⎪
[CHS]      ⎪
[x⇄y]      ⎪
[g] [x²]   ⎪
[+]        ⎭
[CHS]
```

These sections of program memory are identical.

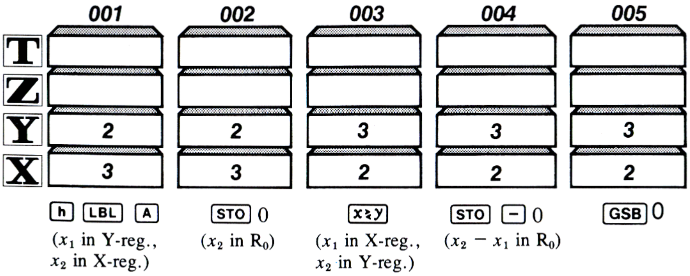
```
            [x⇄y]
          ⎧ [ENTER↑]
          ⎪ [CHS]
          ⎪ [g] [eˣ]
          ⎪ [h] [LST x]
          ⎪ [g] [x²]
Calculates f(x₂) . ⎨ [+]
          ⎪ [f] [LN]
          ⎪ [CHS]
          ⎪ [x⇄y]
          ⎪ [g] [x²]
          ⎩ [+]
            [+]
            [RCL] 0
            [÷]
            [h] [RTN]
```

Since the program section for calculating $f(x_2)$ contains a large portion of program memory identical to the section for calculating $f(x_1)$, you can simply create a subroutine that will execute this section of instructions. The subroutine is then called up and executed in calculating both $f(x_2)$ and $f(x_1)$.

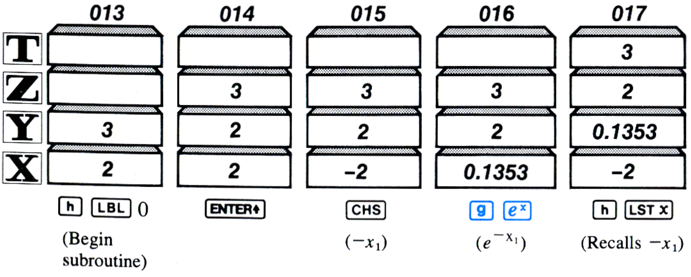


```
001  h LBL A          013  h LBL 0
002  STO 0            014  ENTER↑
003  x⇄y              015  CHS
004  STO − 0          016  g e^x
005  GSB 0            017  h LST x
006  CHS              018  g x²
007  x⇄y              019  +
008  GSB 0            020  f LN
009  +                021  CHS
010  RCL 0            022  x⇄y
011  ÷                023  g x²
012  h RTN            024  +
                      025  h RTN
```

With the modified program, when you press A with x_1 in the Y-register and x_2 in the displayed X-register, execution begins with the h LBL A instruction in line 001. When the GSB 0 instruction in line 005 is encountered, execution transfers to the h LBL 0 instruction in line 013 and calculates the quantity $f(x_1)$. If for example, we used a value of 2 for x_1 and a value of 3 for x_2, here is an illustration of what would be happening in the stack as the average slope of $f(x)$ was calculated.



	001	002	003	004	005
T					
Z					
Y	2	2	3	3	3
X	3	3	2	2	2
	h LBL A	STO 0	x⇄y	STO − 0	GSB 0
	(x_1 in Y-reg., x_2 in X-reg.)	(x_2 in R_0)	(x_1 in X-reg., x_2 in Y-reg.)	($x_2 - x_1$ in R_0)	



	013	014	015	016	017
T					3
Z		3	3	3	2
Y	3	2	2	2	0.1353
X	2	2	−2	0.1353	−2
	h LBL 0	ENTER↑	CHS	g e^x	h LST x
	(Begin subroutine)		($-x_1$)	(e^{-x_1})	(Recalls $-x_1$)

	018	019	020	021	022
T	3	3	3	3	3
Z	2	3	3	3	3
Y	0.1353	2	2	2	−1.4196
X	4	4.1353	1.4196	−1.4196	2
	$\boxed{\text{g}}\ \boxed{x^2}$	$\boxed{+}$	$\boxed{\text{f}}\ \boxed{\text{LN}}$	$\boxed{\text{CHS}}$	$\boxed{x \rightleftarrows y}$
	(x_1^2)	($x_1^2 + e^{-x_1}$)	($\ln(x_1^2 + e^{-x_1})$)	($-\ln(x_1^2 + e^{-x_1})$)	

	023	024	025
T	3	3	3
Z	3	3	3
Y	−1.4196	3	3
X	4	2.5804	2.5804
	$\boxed{\text{g}}\ \boxed{x^2}$	$\boxed{+}$	$\boxed{\text{h}}\ \boxed{\text{RTN}}$
	(x_1^2)	($f(x_1)$)	(Return to main program)

From line 025 execution transfers back to the main program and continues with the first line after the last $\boxed{\text{GSB}}$ instruction. When the $\boxed{\text{GSB}}$ 0 instruction in line 008 is encountered, execution again transfers to the $\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 0 instruction in line 013. To continue our illustration:

	006	007	008	013	014
T	3	3	3	3	3
Z	3	3	3	3	−2.5804
Y	3	−2.5804	−2.5804	−2.5804	3
X	−2.5804	3	3	3	3
	$\boxed{\text{CHS}}$	$\boxed{x \rightleftarrows y}$	$\boxed{\text{GSB}}$ 0	$\boxed{\text{h}}\ \boxed{\text{LBL}}$ 0	$\boxed{\text{ENTER}\uparrow}$
	($-f(x_1)$)	($-f(x_1)$ saved in stack.)		(Begin subroutine)	

	015	016	017	018	019
T	3	3	-2.5804	-2.5804	-2.5804
Z	-2.5804	-2.5804	3	3	-2.5804
Y	3	3	0.0498	0.0498	3
X	-3	0.0498	-3	9	9.0498
	CHS	g e^x	h LST x	g x^2	+
	$(-x_2)$	(e^{-x_2})	(Recalls $-x_2$)	(x_2^2)	$(x_2^2 + e^{-x_2})$

	020	021	022	023	024
T	-2.5804	-2.5804	-2.5804	-2.5804	-2.5804
Z	-2.5804	-2.5804	-2.5804	-2.5804	-2.5804
Y	3	3	-2.2027	-2.2027	-2.5804
X	2.2027	-2.2027	3	9	6.7973
	f LN	CHS	x≷y	g x^2	+
	$(\ln(x_2^2 + e^{-x_2}))$	$(-\ln(x_2^2 + e^{-x_2}))$	(x_2)	(x_2^2)	$(f(x_2))$

	025
T	-2.5804
Z	-2.5804
Y	-2.5804
X	6.7973
	h RTN
	(Return to main program)

After the calculator passes through the subroutine under $\boxed{\text{LBL}}$ 0 a second time to compute $f(x_2)$, the $\boxed{\text{h}}$ $\boxed{\text{RTN}}$ instruction at line 025 causes execution to return to the first instruction in the main program after the last $\boxed{\text{GSB}}$ 0 instruction. $f(x_2)$ is in the X-register; $-f(x_1)$ is in the Y-, Z-, and T-registers.

	009	010	011	012
T	–2.5804	–2.5804	–2.5804	–2.5804
Z	–2.5804	–2.5804	–2.5804	–2.5804
Y	–2.5804	4.2168	–2.5804	–2.5804
X	4.2168	1	4.2168	4.2168
	$\boxed{+}$	$\boxed{\text{RCL}}$ 0	$\boxed{\div}$	$\boxed{\text{h}}$ $\boxed{\text{RTN}}$
	$(f(x_2) - f(x_1))$	$(x_2 - x_1)$	$(f(x_2) - f(x_1) \div (x_2 - x_1))$	(End of program)

When calculation halts, the average slope of $f(x)$ between x_1 and x_2 appears in the display. Extra copies of $-f(x_1)$ in the Y-, Z-, and T-registers are ignored.

Now key in the program and try the problems on the next page. Slide the PRGM-RUN switch to PRGM .

Keystrokes	Display
$\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$	000–
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$	001– 25, 13, 11
$\boxed{\text{STO}}$ 0	002– 23 0
$\boxed{x \gtrless y}$	003– 21
$\boxed{\text{STO}}$ $\boxed{-}$ 0	004– 23, 41, 0
$\boxed{\text{GSB}}$ 0	005– 13 0
$\boxed{\text{CHS}}$	006– 32
$\boxed{x \gtrless y}$	007– 21
$\boxed{\text{GSB}}$ 0	008– 13 0
$\boxed{+}$	009– 51
$\boxed{\text{RCL}}$ 0	010– 24 0
$\boxed{\div}$	011– 71
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	012– 25 12
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 0	013– 25, 13, 0
$\boxed{\text{ENTER}\uparrow}$	014– 31
$\boxed{\text{CHS}}$	015– 32
$\boxed{\text{g}}$ $\boxed{e^x}$	016– 15 1
$\boxed{\text{h}}$ $\boxed{\text{LST x}}$	017– 25 0
$\boxed{\text{g}}$ $\boxed{x^2}$	018– 15 3
$\boxed{+}$	019– 51
$\boxed{\text{f}}$ $\boxed{\text{LN}}$	020– 14 1
$\boxed{\text{CHS}}$	021– 32

```
[x≷y]          022-       21
[g] [x²]       023-    15  3
[+]            024-       51
[h] [RTN]      025-    25 12
```

Slide the PRGM-RUN switch to ▮▮▮▮[|||||]RUN . Now find the average slope of $f(x)$ between the following pairs of points: (0, 0.5), (0.55, 1.15), (1.25, 1.75).

Answers: 0.8097, 0.6623, 1.8804.

Subroutine Usage

Subroutines give you extreme versatility in programming. A subroutine can contain a loop, or it can be executed as part of a loop. Another common and space-saving trick is to use the same routine as a subroutine and as part of the main program.

Example: The program below simulates the throwing of a pair of dice, pausing to display first the value of one die (an integer from 1 to 6) and then pausing to display the value of the second die (another integer from 1 to 6). Finally the values of the two dice are added together to give the total value.

The "heart" of the program is a random number generator (actually a pseudorandom number generator) that is executed first as a subroutine and then as part of the main program. When you key in a first number, called a "seed," and press [A], the digit for the first die is generated and displayed using the [h] [LBL] 2 routine as a subroutine. Then the digit for the second die is generated using the same routine as part of the main program. The program then uses the generated number as a new seed for successive "throws" of the dice.

To key in the program:

Set the calculator to PRGM mode.

Keystrokes	Display	
f CLEAR PRGM	000-	
GTO 1	001- 22 1	
h LBL A	002- 25, 13, 11	
STO 0	003- 23 0	
h LBL 1	004- 25, 13, 1	
0	005- 0	
STO 1	006- 23 1	
GSB 2	007- 13 2	h LBL 2 executed first as a subroutine.
h LBL 2	008- 25, 13, 2	
RCL 0	009- 24 0	
9	010- 9	
9	011- 9	
7	012- 7	
×	013- 61	
h FRAC	014- 25 33	
STO 0	015- 23 0	
6	016- 6	
×	017- 61	
1	018- 1	h LBL 2 then executed as the remainder of the main program.
+	019- 51	
h INT	020- 25 32	
⁝ FIX 0	021- 14, 11, 0	
h PSE	022- 25 74	
STO + 1	023- 23, 51, 1	
RCL 1	024- 24 1	
h RTN	025- 25 12	Transfers execution to line 008 when LBL 2 executed as a subroutine; to line 000 when LBL 2 executed as the remainder of the main program.

Now set the calculator to ▮▮▮▮ RUN mode and "roll" the dice. To roll the dice, key in the initial decimal "seed" (that is, $0 < n < 1$). Then press [A]. The calculator will display first the number rolled by the first die, then the number rolled by the second, and finally, when the program stops, you can see the total number rolled by the dice. To make another roll, press [R/S]. The program uses the last number as a new seed for the roll.

You can play a game with your friends using the "dice." If your first "roll" is 7 or 11, you win. If it is another number, that number becomes your "point." You then keep "rolling" (pressing [R/S]) until the dice again total your point (you win) or you roll a 7 or 11 (you lose). To run the program:

Keystrokes	Display	
.2315478	*0.2315478*	The seed.
[A]	*10.*	Your point is 10.
[R/S]	*8.*	You missed your point.
[R/S]	*5.*	Missed it again.
[R/S]	*7.*	Whoops! You lose.

Now try it again using the last number as the new seed.

Keystrokes	Display	
[R/S]	*8.*	Your point is 8.
[R/S]	*8.*	Congratulations! You win.

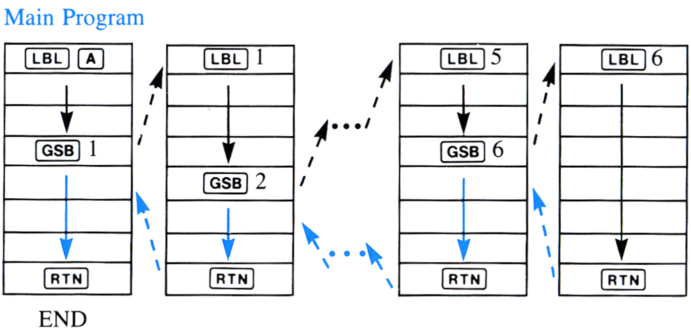
Before you continue, reset the display to four decimal places.

Keystrokes	Display
[f] [FIX] 4	*8.0000*

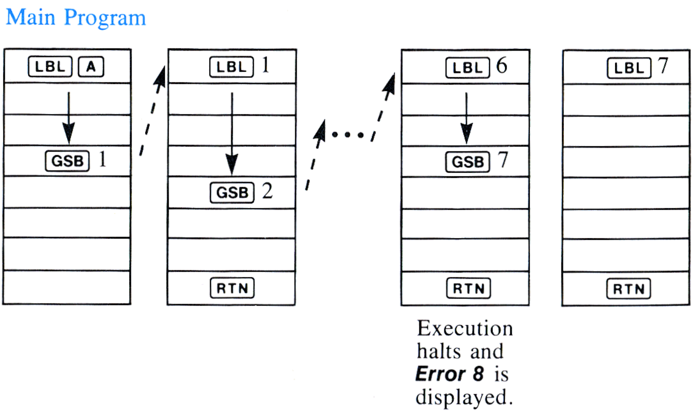
Subroutine Limits

A subroutine can call up another subroutine, and that subroutine can call up yet another subroutine. Subroutine branching is limited only by the number of *returns* that can be held pending by the calculator. Six subroutine returns can be held pending at any one time in the HP-34C.

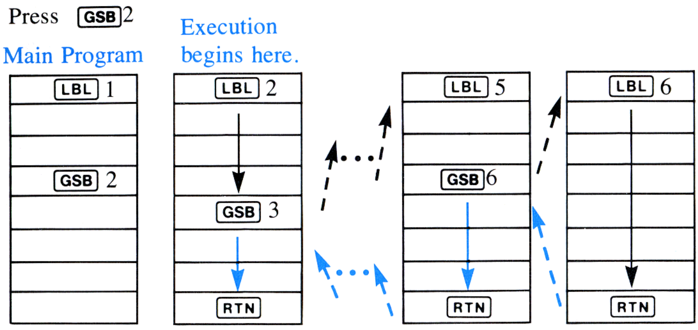
The diagram below should make this more clear.



The calculator can return back to the main program from subroutines that are six deep, as shown. However, if you attempt to call up subroutines that are seven deep, the calculator will halt and display ***Error 8*** when it encounters the instruction attempting to call the seventh subroutine level.



Naturally, the calculator can execute non-subroutine [RTN] instructions (transfer execution to line 000 and halt) any number of times. Also, if you press [GTO] or [GSB] with [A], [B], or 0 through 9 from the keyboard, any pending [RTN] instructions are forgotten by the calculator.



Note that in PRGM mode, single-step execution of a program containing subroutines follows the same order of execution as in a running program.

Using [h] [RTN] at the End of Occupied Program Memory

The programming examples in your *HP-34C Owner's Handbook and Programming Guide* include an [h] [RTN] as the last line in occupied program memory. This is done both to clearly indicate the ends of programs and to illustrate how [RTN] affects program execution. However, you can omit [h] [RTN] where it occurs as the last instruction in occupied program memory without affecting program execution. Why? Whenever the last instruction in program memory is not [h] [RTN], program execution performs just as if [h] [RTN] existed immediately following the last instruction you keyed in. In other words, when program execution encounters the end of occupied memory without finding an [h] [RTN] instruction:

1. If in a subroutine, execution returns to the first line after the last GSB instruction and resumes.

2. If not in a subroutine, execution returns to line 000 and halts.

If the last line in occupied memory contains a GSB instruction, the calculator executes the indicated subroutine, returns to line 000, and halts.

Notice that GTO and GSB instructions always cause the calculator to search *forward* in program memory for the specified label. This feature often allows you to write a program in such a way that it uses a given label more than once.

Example: The following program to calculate the value of the expression $\sqrt{x^2 + y^2 + z^2 + t^2}$ uses LBL A to identify both the beginning of the program and a subroutine within the program. The program is executed by placing the variables x, y, z, and t in the stack and pressing A.

Slide the PRGM-RUN switch to PRGM ▯▮ and key in the following program.

Keystrokes	Display
f CLEAR PRGM	000–
h LBL A	001– 25, 13, 11
g x^2	002– 15 3
GSB A	003– 13 11
GSB A	004– 13 11
GSB A	005– 13 11
f $\sqrt{x}$	006– 14 3
h RTN	007– 25 12
h LBL A	008– 25, 13, 11
x⪤y	009– 21
g x^2	010– 15 3
+	011– 51
h RTN	012– 25 12

Slide the PRGM-RUN switch to ▮▥ RUN and key in the following set of variables:

$$x = 4.3,\ y = 7.9,\ z = 1.3,\ t = 8.0$$

Keystrokes	Display
8 $\boxed{\text{ENTER}\uparrow}$	***8.000***
1.3 $\boxed{\text{ENTER}\uparrow}$	***1.3000***
7.9 $\boxed{\text{ENTER}\uparrow}$ 4.3 $\boxed{\text{A}}$	***12.1074***

Advanced Programming

Controlling the I-Register

The I-register is one of the most powerful programming tools available on your HP-34C. In addition to serving as a register for the simple storage and recall of data, the I-register can also be used in conjunction with other instructions to perform the following:

- Increment or decrement a specified value from the current value in I for loop control or other functions.
- Indirectly control the storage register address of STO, RCL, and storage register arithmetic.
- Indirectly control the label address of GTO and GSB.
- Indirectly control the number of digits displayed by the FIX, SCI, and ENG modes.
- Transfer execution to any line of occupied program memory.

Storing a Number in the I-Register

To store a number in the I-register, you use the key sequence STO f I. For example, to store the number 7 in the I-register:

Ensure that the PRGM-RUN switch is set to RUN .

Keystrokes	Display
7 STO f I	7.0000

To recall a number from the I-register into the displayed X-register, you use the key sequence RCL f I.

Keystrokes	Display	
CLX	0.0000	
RCL f I	7.0000	A copy of the number stored in I is recalled.

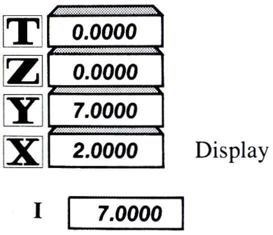
Exchanging X and I

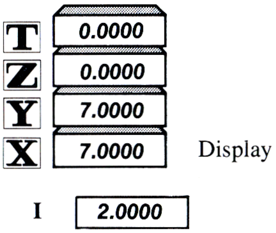
In a manner similar to the $\boxed{x\gtrless y}$ operation, the $\boxed{f}$ $\boxed{x\gtrless I}$ (X exchange I) operation exchanges the contents of the displayed X-register with those of the I-register. For example, key the number 2 into the displayed X-register and exchange the contents of the X-register with the value you stored in the I-register in the previous example.

Keystrokes	Display	
2	**2.**	
$\boxed{f}$ $\boxed{x\gtrless I}$	**7.0000**	Contents of X-register and I-register exchanged.

When you pressed $\boxed{x\gtrless I}$, the contents of the stack and the I-register were changed…

…from this… **…to this.**





To restore the X-register and I-register contents to their original positions:

Keystrokes	Display
$\boxed{f}$ $\boxed{x\gtrless I}$	**2.0000**

Incrementing and Decrementing the I-Register

Another way of altering the contents of the I-register, and one that is most useful in programming, is through the ISG (increment, then skip if greater) and DSE (decrement, then skip if less than or equal) functions. Both contain internal counters that allow you to control the execution of a loop, as well as the sequential addressing operations covered later in this section.

The ISG and DSE functions use a number that is stored in the I-register and interpreted in a special way. The number is called a *loop control value*. The usual format is:

nnnnn.xxxyy

A loop control value is interpreted as three separate integers, where:

±**nnnnn**	is the current counter value,
xxx	is the counter test value, and
yy	is the increment or decrement value.

The **nnnnn** portion of the number tells your HP-34C that you wish to count the number of passes through the loop beginning with that number. If you do not specify an **nnnnn** value, the HP-34C assumes you wish to begin counting at zero. An **nnnnn** value can be specified as one to five digits.

The **xxx** portion of the number tells the HP-34C that you wish to stop the counting at that number. The **xxx** value must always be specified as a three-digit number (e.g., an **xxx** value of 10 would be specified as 010).

The **yy** portion of the loop control number tells the calculator how you wish to count. Current counter value **nnnnn** is incremented or decremented by the value of **yy.** If you do not specify a **yy** value, the HP-34C automatically assumes you wish to count by ones (**yy** default = 01). A specified ***yy*** value must be two digits (e.g., 02, 03, 55).

Increment, Then Skip if Greater. Each time ISG is executed, it first increments ***nnnnn*** by ***yy.*** It then tests to see if ***nnnnn*** is greater than **xxx.** If it is, the HP-34C skips the next line in the program.

So, with the loop control value 100.20001 in the I-register, the ISG instruction would begin counting up from 100. Each time the program executed ISG, the **nnnnn** portion of the loop control value would be incremented by 1.

Contents of the I-register = 100.20001
Execution of ISG would:
 Start counting up from 100.
 Increment **nnnnn** by 1.
 Test to see if **nnnnn** is greater
 than 200.

After one execution or pass through the loop containing ISG, the I-register would contain 101.20001. After 10 executions or passes through the loop, the I-register would contain 110.20001. Each time ISG increments, it then checks to see if the current counter value **nnnnn** is greater than 200 (**xxx**). When **nnnnn** is greater than 200, program execution skips the next line of program memory following the ISG instruction. You will see how skipping the next line in the program is useful in a moment.

Decrement, then Skip if Equal (or Less Than). Each time DSE is executed, it first decrements **nnnnn** by **yy.** It then tests to see if **nnnnn** is equal to (or less than) **xxx.** If it is, the HP-34C skips the next line in the program.

So, with the number 100.01001 in the I-register, the DSE instruction would begin counting down from 100. Each time the program executed DSE, the **nnnnn** portion of the loop control value would be decremented by 1.

Contents of the I-register = 100.01001
Execution of DSE would:
- Start counting down from 100.
- Decrement by 1.
- Test to see if **xxx** was equal to (or less than) 10.

After one execution or pass through the loop, the I-register would contain 99.01001. After 10 executions or passes through the loop, the I-register would contain 90.01001. Each time DSE decrements, it then checks to see if the counter value **nnnnn** is equal to or less than 010 **(xxx).** When **nnnnn** is equal to or less than 010 (**xxx**), the calculator skips the next line of the program.

Example: Here is a program that illustrates how ISG works. It contains a loop that pauses to display the current value in the I-register and uses ISG to control the number of passes through the loop and the value of the squared number. The program generates a table of squares of even numbers from 2 through 50.

Slide the PRGM-RUN switch to PRGM ▮▮ and key in the following program.

Keystrokes	Display	
f CLEAR PRGM	000–	
h LBL A	001– 25, 13, 11	Program label
f FIX 5	002– 14, 11, 5	
2	003– 2	Current counter value (**nnnnn**).
•	004– 73	
0	005– 0	Counter test value (**xxx**).
5	006– 5	
0	007– 0	
0	008– 0	Increment value (**yy**).
2	009– 2	
STO f I	010– 23, 14, 23	Store loop control value in 1.
h LBL 1	011– 25, 13, 1	Begin the loop.
RCL f I	012– 24, 14, 23	Recall the number in I.
h INT	013– 25 32	Take the integer portion.
h PSE	014– 25 74	Pause to display the integer.
g x^2	015– 15 3	Square the number.
h PSE	016– 25 74	Display the square of the number.
g ISG	017– 15 24	Increment I by 2 and check to see that the counter is not greater than the final number (50). If the counter is greater than the final number, skip the next line in the program.
GTO 1	018– 22 1	Loop back to label 1.
h RTN	019– 25 12	Halts the program.

Now run the program:

Slide the PRGM-RUN switch to RUN and press $\boxed{A}$.

Keystrokes	**Display**	
$\boxed{A}$	**2.00000**	When the HP-34C begins executing, it first pauses to display the number to be squared, then pauses to display the square of the number. When the loop counter increments beyond 50, the program halts.
	4.00000	
	4.00000	
	16.00000	
	.	
	.	
	.	
	50.00000	
	2,500.00000	

Here is what happens when you run the above program.

1. Under label $\boxed{A}$, the number 2.05002 is stored in the I-register as the loop control value. It is in the counter format: i.e.,

nnnnn	**xxx**	**yy**
(0000)2	050	02
Current counter Value	Test Value	Increment Value

2. Under label 1, the following sequence occurs:

After 2 and 4 (the square of 2) are displayed, the current counter value in I, 00002 (**nnnnn**), is incremented by the increment value 02 (**yy**). The new number in the I-register is 4.05002, which is interpreted by your calculator as:

nnnnn	**xxx**	**yy**
(0000)4	050	02
Current Counter Value	Test Value	Increment Value

The new counter value is then compared to the test value 050 (**xxx**). As the counter value has not exceeded the test value, the calculator proceeds to the next line, $\boxed{\text{GTO}}$ 1, and the process is repeated with the new number.

3. After 25 even-numbers (2-50) and their squares are displayed, the current counter value finally increments beyond 50. This causes the calculator to skip one line after the $\boxed{\text{g}}$ $\boxed{\text{ISG}}$ at line 17. As a result, the $\boxed{\text{GTO}}$ 1 command at line 18 is bypassed and the $\boxed{\text{RTN}}$ command at line 19 is executed, causing the calculator to return to line 000 and halt.

After running the program, press $\boxed{\text{RCL}}$ $\boxed{\text{f}}$ $\boxed{\text{I}}$. The recalled I-register value in your display should now look like this:

$$\underbrace{52}_{\substack{\text{Current}\\ \text{Counter}\\ \text{Value}\\ (\textbf{nnnnn})}}.\underbrace{050}_{\substack{\text{Test}\\ \text{Value}\\ (\textbf{xxx})}}\underbrace{02}_{\substack{\text{Increment}\\ \text{Value}\\ (\textbf{yy})}}$$

Now let's add a second program which uses your HP-34C's $\boxed{\text{DSE}}$ function. Remember, the **nnnnn.xxxyy** format is the same as for $\boxed{\text{ISG}}$. You will, however, be decrementing the current counter value instead of incrementing it.

The island of Manhattan was sold in the year 1624 for $24. The following program shows a simplified method to calculate growth of the original amount if it had been placed in a bank account drawing 6% annual interest. The number of years for which you want to calculate growth is stored in the I-register as a loop control value. The $\boxed{\text{DSE}}$ instruction is then used to keep track of the number of iterations through the loop.

Slide the PRGM-RUN switch to PRGM ▥█ . Executing the $\boxed{\text{RTN}}$ instruction in line 019 of the previous program returned your calculator to line 000. To add the following program to the end of currently occupied program memory, press $\boxed{\text{h}}$ $\boxed{\text{BST}}$ (or $\boxed{\text{GTO}}$ $\boxed{\cdot}$ 019) to return to line 019.

Keystrokes	Display	
$\boxed{\text{h}}$ $\boxed{\text{BST}}$	*019– 25 12*	Last line of previous program.
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{B}}$	*020– 25, 13, 12*	New program label.
$\boxed{\text{f}}$ $\boxed{\text{FIX}}$ 2	*021– 14, 11, 2*	
$\boxed{\text{STO}}$ $\boxed{\text{f}}$ $\boxed{\text{I}}$	*022– 23, 14, 23*	Stores user-input loop control value **nnnnn.xxxyy** in the I-register.
1	*023– 1*	Initial year.
6	*024– 6*	
2	*025– 2*	
4	*026– 4*	
$\boxed{+}$	*027– 51*	Final year.
$\boxed{\text{STO}}$ 0	*028– 23 0*	Stores final year.
2	*029– 2*	Initial dollar amount.
4	*030– 4*	
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 2	*031– 25, 13, 2*	Begins the loop.
1	*032– 1*	Calculates annual growth.
0	*033– 0*	
6	*034– 6*	
$\boxed{\text{h}}$ $\boxed{\%}$	*035– 25 41*	
$\boxed{\text{g}}$ $\boxed{\text{DSE}}$	*036– 15 23*	Decrements the current counter value **nnnnn** and compares with the counter test value **xxx**.
$\boxed{\text{GTO}}$ 2	*037– 22 2*	If **nnnnn>xxx**, returns to $\boxed{\text{LBL}}$ 2.
$\boxed{\text{RCL}}$ 0	*038– 24 0*	If **nnnnn≤xxx**, displays final year, final growth value, and halts.
$\boxed{\text{h}}$ $\boxed{\text{PSE}}$	*039– 25 74*	
$\boxed{x \gtrless y}$	*040– 21*	
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	*041– 25 12*	

Slide the PRGM-RUN switch to RUN and key in the number of years (loop control value) for which you want to see the accumulated amount. Press B to store your input value in the I-register and to run the program.

Keystrokes	Display	
5	**5.**	Loop control value; **nnnnn** = 5, **xxx** = 000, **yy** = 00 (defaults to 01 internally).
B	***32.12***	After five years, in 1629, the account would have been worth $32.12.
15	***15.***	Loop control value; **nnn** = 15, **xxx** = 000, **yy** = 00 (defaults to 01 internally).
B	***57.52***	After 15 years, in 1639, the account would have been worth $57.52.

How it works: When you key in the number of years and press B your entry is stored in the I-register and becomes the loop control value (**nnnnn.xxxyy**).

nnnnn	**xxx**	**yy**
(0000)5	000	00
Current Counter Value	Counter Test Value	Decrement Value (Defaults to 01 internally.)

(Notice that when the test value is 000 and the increment or decrement value is 01, it is not necessary to enter them.)

The loop control value is then added to the initial year. This sum is the final year and is stored in R_0 for later recall. The initial dollar amount is then entered. Each time through the loop the dollar amount is increased by 6%. The DSE instruction then subtracts 1 from the I-register. If the loop control value in I is not then zero, execution returns to LBL 2 and the loop is executed again.

When the loop control value in the I-register is decremented to zero (**nnnnn**=**xxx**), execution bypasses the GTO 2 instruction at line 37 and resumes with the RCL 0 instruction at line 38. The final year and dollar value then appear in succession and the program halts.

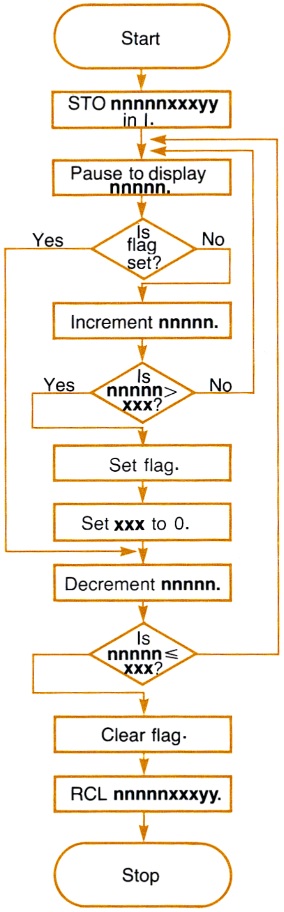
ISG and DSE Limits

Note that ISG and DSE can be used to increment and decrement any number that the HP-34C can display. However, the decimal portion of the loop control value will be affected by current counter values exceeding the five-digit **nnnnn** value.

For example, the number 99,950.50055, when incremented using ISG would become 100,005.5006. The initial number was incremented by 55. But since the new number 100,005.50055, cannot be fully displayed, the decimal portion of the number was rounded. As the calculator assumes a two-digit number for the increment value (**yy**), the next increment would be by 60, not 55. And when the number becomes 999,945.5006, the next number would be 1,000,005.501, thus rounding the decimal portion of the number again. Since no increment value **yy** is present, the next increment would default to 01 instead of remaining at 60.

Problem:

1. Write a program that will count from zero up to a limit using the ISG function, and then, in the same program, count back down to zero using the DSE function. Use the flowchart on the following page to help you.


Start
STO **nnnnnxxxyy** in I.
Pause to display **nnnnn.**
Is flag set?
Yes
No
Increment **nnnnn.**
Is **nnnnn** > **xxx**?
Yes
No
Set flag.
Set **xxx** to 0.
Decrement **nnnnn.**
Is **nnnnn** ≤ **xxx**?
Clear flag.
RCL **nnnnnxxxyy.**
Stop

Using The I-Register For Display, Storage Register, and Program Control

You have seen how the value in the I-register can be altered using [STO], [x⪤I], [ISG], and [DSE] operations. But the value contained in the I-register can also be used to control display, storage register, branching, and subroutine operations. First, let's get a brief overview of these operations. Then we'll examine each one in detail.

[DSP I] *(display I)* uses a number stored in the I-register to specify the number of decimal places appearing in the display.

[x⪤(i)] *(X exchange indirect)* exchanges the contents of the displayed X-register with the contents of the available storage register addressed by the absolute value of the number in the I-register.

[STO] [f] [(i)] *(store indirect)* stores the value that is in the display in the storage register addressed by the absolute value of the number currently in the I-register.

[RCL] [f] [(i)] *(recall indirect)* recalls the contents of the storage register addressed by the absolute value of the number currently in the I-register.

[STO] ([+], [×], [−], or [÷]) [(i)] *(indirect storage register arithmetic)* performs storage register arithmetic on the contents of the storage register addressed by the absolute value of the number currently in the I-register.

[GTO] [f] [I] *(go to label or line I)* with a *positive* number in the I-register transfers execution of a running program sequentially downward in program memory to the next label specified by the number currently in I. With a *negative* number in the I-register, execution transfers to the occupied *line* number specified by the absolute value of the number currently in I.

[GSB] [f] [I] *(go to label or line I subroutine)* with a *positive* number in the I-register transfers execution of a running program sequentially downward in program memory to the next *label* specified by the number currently in I. With a *negative* number in the I-register, execution transfers to the occupied *line number* specified by the absolute value of the number currently in I. In both cases, when a [RTN] is then encountered, execution transfers back to the line following the [GSB] instruction, and continues.

When executing any one of the above operations, if the number in the I-register is inappropriate for that operation, the display will show an ***Error*** message. Also, when using a number in I for display, storage register, or program control, remember that the calculator uses only the integer portion of the number in I. Thus, 12.99041276 stored in the I-register retains its full value there, but when used to control any of the above operations it is read as 12 by the calculator.

You can already see that using the I-register in conjunction with other functions gives you a tremendous amount of computing power and exceptional programming control. Now let's have a closer look at these operations.

I-Register Display Control

You can use a number in the I-register to control the number of decimal places appearing in the display. When [h] [DSP I] is performed, the display is seen rounded to the number of decimal places specified by the current value contained in the I-register. (The display is *seen* rounded, but of course, the calculator maintains its full accuracy, 10 digits multiplied by 10 raised to a two-digit exponent, internally.) The above operation is most useful as part of a program, but it can also be executed manually from the keyboard. For example, execute the following in RUN mode.

Keystrokes	Display	
[CLx] [f] [FIX] 4	**0.0000**	Clears display; normal FIX display.
[STO] [f] [I]	**0.0000**	Insures that zero is in the I-register.
9.123456789	**9.123456789**	
[h] [DSP I]	**9.**	FIX display specified by the zero value in the I-register.
[g] [ISG]	**9.**	Increments value in I-register to 1.
[h] [DSP I]	**9.1**	FIX display specified by the the value in the I-register.
[g] [ISG]	**9.1**	Increments value in I-register to 2.
[h] [DSP I]	**9.12**	FIX display specified by the value in the I-register.

Example: The following program pauses and displays an example of FIX display format for each possible decimal place. It utilizes a loop containing a DSE instruction to automatically change the number of decimal places.

Slide the PRGM-RUN switch to PRGM ▯■ and key in the following program.

Keystrokes	Display
f CLEAR PRGM	000-
h LBL A	001- 25, 13, 11
9	002- 9
STO f I	003- 23, 14, 23
h LBL 0	004- 25, 13, 0
h DSP I	005- 25 11
RCL f I	006- 24, 14, 23
h PSE	007- 25 74
g DSE	008- 15 23
GTO 0	009- 22 0
g $x>0$	010- 15 51
GTO 0	011- 22 0
h RTN	012- 25 12

To display fixed point notation for all possible decimal places on your HP-34C.

Slide PRGM-RUN switch to ■▯ RUN .

Keystrokes	Display
A	9.000000000
	8.00000000
	7.0000000
	6.000000
	5.00000
	4.0000
	3.000
	2.00
	1.0
	0.
	0.

To display scientific or engineering notation for all possible places, replace the 9 at line 002 with a 6 and shift the calculator to SCI or ENG mode by pressing [f] [SCI] or [f] [ENG] and any digit 0-7.* Then press [A] as you did in the above example.

Slide the PRGM-RUN switch PRGM ▮▮ to PRGM.

Keystrokes	Display
[GTO] [•] 002	*002– 9*
[h] [DEL]	*001– 25, 13, 11*
6	*002– 6*

Slide the PRGM-RUN switch to ▮▮ RUN .

Keystrokes	Display	
[f] [SCI] 4	*0.0000 00*	Normal [SCI] display.
or		
[f] [ENG] 4	*0.0000 00*	Normal [ENG] display.
[A]	*6.000000 00*	
	5.00000 00	
	4.0000 00	
	3.000 00	
	2.00 00	
	1.0 00	
	0. 00	
	0. 00	

If any number less than 0 is stored in the I-register, executing [h] [DSP I] results in the same number of digits in the display as when you execute [h] [DSP I] with 0 in the I-register.† If a number greater than 9 is stored in the I-register, executing [h] [DSP I] results in the same number of digits in the display as when you execute [h] [DSP I] with 9 in the I-register. Note that in SCI and ENG modes any number greater than 6 in the I-register results in a maximum of 6 digits and a 2-digit exponent

* In PRGM mode, pressing [f] [SCI] or [f] [ENG] followed by 8 or 9 automatically results in an [f] [SCI] 7 or [f] [ENG] 7 in program memory.

† During execution of [$\int_y^x$] only, a number –6 through +9 in the I-register is used by [DSP I] as an automatic parameter for [$\int_y^x$] calculations (more on [$\int_y^x$] in section 9).

appearing to the right of the decimal. (Remember, however, that SCI or ENG 7 rounds the display to one more digit than does SCI or ENG 6.)

Execute the following:

Keystrokes	Display	
f FIX 4	**0.0000**	Normal FIX display.
1.999999999	**1.999999999**	
STO f I	**2.0000**	Display rounds to last display format command.
h DSP I	**2.0**	Only the integer portion of the value in I is read by DSP I.
.9852	**0.9852**	
STO f I	**1.0**	Display rounds to last format command.
h DSP I	**1.**	A value of <1 brings the same result as a value of 0.
19	**19.**	
STO f I	**19.**	
h DSP I	**19.00000000**	With 2 digits to the left of the decimal occupied, a value $>$ 9 stored in I brings the same result as a value of 8 or 9.
CHS STO f I	**−19.00000000**	Stores a negative number in I.
h DSP I	**−19.**	A negative number stored in I brings the same result as a positive number <1.
f SCI 4	**−1.9000 01**	Normal SCI display.
1.1111119 ENTER↑	**1.1111 00**	
7 STO f I x⇄y	**1.1111 00**	
h DSP I	**1.111111 00**	Display rounded to 7 decimal places.
6 STO f I x⇄y	**1.111111**	
h DSP I	**1.111112 00**	Display rounded to 6 decimal places.

Exchanging X and (i)

Using $\boxed{x \gtrless (i)}$ you can exchange the contents of the displayed X-register with those of any available storage register indirectly addressed by the absolute value of any number $-21 < n < 21$ in the I-register. The integers from 0 through ± 9 address storage registers R_0 through R_9. The integers from ± 10 through ± 19 address registers $R_{.0}$ through $R_{.9}$. With the number ± 20 in the I-register, $\boxed{x \gtrless (i)}$ addresses the I-register itself!

The following diagram illustrates these addresses more clearly:

	(i) Address		(i) Address
R_0	0	$R_{.0}$	10
R_1	1	$R_{.1}$	11
R_2	2	$R_{.2}$	12
R_3	3	$R_{.3}$	13
R_4	4	$R_{.4}$	14
R_5	5	$R_{.5}$	15
R_6	6	$R_{.6}$	16
R_7	7	$R_{.7}$	17
R_8	8	$R_{.8}$	18
R_9	9	$R_{.9}$	19
		I	20

Before proceeding, set the display to FIX 4 and clear both the displayed X-register and all storage registers.

Keystrokes	Display
$\boxed{f}$ $\boxed{FIX}$ 4 $\boxed{CLX}$	*0.0000*
$\boxed{f}$ CLEAR $\boxed{REG}$	*0.0000*

Now try the following examples using $\boxed{x \gtrless (i)}$ to store 1.234 in registers R_3, $R_{.5}$, and I.

Keystrokes	Display	
3 f x⇄I	0.0000	Exchanges contents of displayed X-register and I-register.
1.2345 h x⇄(i)	0.0000	Exchanges contents of displayed X-register and R_3, using the integer 3 in I for an address.
RCL 3	1.2345	Recalls a copy of the contents of R_3.
15 f x⇄I	3.0000	Exchanges contents of displayed X-register and I.
x⇄y	1.2345	Exchanges contents of displayed X-register and Y-register.
h x⇄(i)	0.0000	Exchanges contents of displayed X-register and $R_{.5}$ using the integer 15 in I as an address.
RCL • 5	1.2345	Recalls a copy of the contents of $R_{.5}$.
f CLEAR REG	1.2345	Clears the contents of all storage registers to 0.
15.3974 CHS	-15.3974	
f x⇄I	0.0000	Exchanges the contents of the displayed X-register and I.
x⇄y	1.2345	Exchanges the contents of the displayed X-register and the Y-register.
h x⇄(i)	0.0000	Exchanges contents of displayed X-register and $R_{.5}$ using the integer portion of the absolute value of −15.3974 stored in I as an address.
RCL • 5	1.2345	Recalls a copy of the contents of $R_{.5}$

Keystrokes	Display	
20 [f] [x⇄I]	-15.3974	Exchanges the contents of the displayed X-register and I.
[x⇄y]	1.2345	Exchanges the contents of the displayed X-register and the Y-register.
[h] [x⇄(i)]	20.0000	Exchanges the contents of the displayed X-register and I, using the integer stored in I as an address.
[RCL] [f] [I]	1.2345	Recalls a copy of the contents of I.
[f] CLEAR [REG]	1.2345	Clears the contents of all storage registers to 0.
[CLX] [ENTER↑] [ENTER↑] [ENTER↑]	0.0000	Clears all stack registers.

Indirect Store and Recall

Like [x⇄(i)], you can use the I-register to indirectly address all 21 storage registers for [STO] and [RCL] operations. When you press [STO] [f] [(i)], the value in the display is stored in the storage register addressed by the number in the I-register. [RCL] [f] [(i)] addresses the storage registers in a like manner, as do the storage register arithmetic operations [STO] [+] [(i)], [STO] [−] [(i)], [STO] [×] [(i)], and [STO] [÷] [(i)]. (If you have forgotten the normal operation of the storage registers, or of storage register arithmetic, go back and review section 4, Storing and Recalling Numbers, in *Solving Problems With Your Hewlett-Packard Calculator*.)

When using [STO] [f] [(i)], [RCL] [f] [(i)], or any of the storage register arithmetic operations utilizing the [(i)] function, the I-register can contain the same positive or negative values from 0 through 20, as used with [x⇄(i)].

By using the calculator manually, you can easily see how $\boxed{\text{STO}}$ $\boxed{\text{f}}$ $\boxed{\text{(i)}}$ and $\boxed{\text{RCL}}$ $\boxed{\text{f}}$ $\boxed{\text{(i)}}$ are used in conjunction with the I-register to address the different storage registers:

Ensure that the PRGM-RUN switch is set to RUN .

Keystrokes	**Display**	
$\boxed{\text{CLx}}$ $\boxed{\text{f}}$ $\boxed{\text{FIX}}$ 4	***0.0000***	
$\boxed{\text{f}}$ CLEAR $\boxed{\text{REG}}$	***0.0000***	Clears all storage registers, including I, to zero.
5 $\boxed{\text{STO}}$ $\boxed{\text{f}}$ $\boxed{\text{I}}$	***5.0000***	Stores the number 5 in the I-register.
1.23 $\boxed{\text{STO}}$ $\boxed{\text{f}}$ $\boxed{\text{(i)}}$	***1.2300***	Stores the number 1.23 in the storage register addressed by the number in I—that is, storage register R_5.
19 $\boxed{\text{STO}}$ $\boxed{\text{f}}$ $\boxed{\text{I}}$	***19.0000***	Stores the number 19 in the I-register.
85083 $\boxed{\text{STO}}$ $\boxed{\text{f}}$ $\boxed{\text{(i)}}$	***85,083.0000***	Stores the number 85083 in the storage register $R_{.9}$ addressed by the current number 19 in I.
12 $\boxed{\text{STO}}$ $\boxed{\text{f}}$ $\boxed{\text{I}}$	***12.0000***	Stores the number 12 in the I-register.
77 $\boxed{\text{EEX}}$ 43	***77. 43***	
$\boxed{\text{STO}}$ $\boxed{\text{f}}$ $\boxed{\text{(i)}}$	***7.7000 44***	Stores the number 7.7×10^{44} in the storage register addressed by the number in I—that is, in storage register $R_{.2}$.

To recall numbers that are stored in any register, you can use the RCL *(recall)* key followed by the number of the register address. However, when the number currently stored in the I-register addresses the storage register you want, you can recall the contents of that register with RCL f (i).

Keystrokes	Display	
RCL 5	***1.2300***	Contents of storage register R_5 recalled to displayed X-register.
RCL f (i)	***7.7000 44***	Since the I-register still contains the number 12, this operation recalls the contents of storage register $R_{.2}$, which is addressed by the number 12.

By changing the number in the I-register, you change the address specified by STO f (i) or RCL f (i). For example:

Keystrokes	Display	
19 STO f I	***19.0000***	
RCL f (i)	***85,083.0000***	Contents of storage register $R_{.9}$ recalled to displayed X-register.
5 STO f I	***5.0000***	
RCL f (i)	***1.2300***	Contents of storage register R_5 recalled to displayed X-register.

Storage register arithmetic is performed upon the contents of the register addressed by I by using STO + (i), STO − (i), STO × (i), and STO ÷ (i). Notice that it is not necessary to use the f shift function key with these four operations.

Keystrokes	Display	
1 STO + (i)	1.0000	One added to number in storage register (R_5) currently addressed by the I-register.
RCL f (i)	2.2300	Recalls the number stored in R_5.
2 STO × (i)	2.0000	Multiplies the contents of R_5 by 2.
RCL f (i)	4.4600	Recalls the new contents of R_5.
CLx	0.0000	Clears display.
RCL 5	4.4600	Directly recalls the contents of R_5.

Note: When programming, storage register arithmetic commands for register R_0 through R_9 can be keyed in as either direct or indirect storage operations. However, storage register arithmetic commands for registers $R_{.0}$ through $R_{.9}$ and the I-register are implemented using indirect storage operations only.

Naturally, the most effective use of the I-register as an address for STO and RCL is in a program.

Example: The following program uses a loop to place the number representing its address in storage registers R_0 through R_9 and registers $R_{.0}$ through $R_{.9}$. During each iteration through the loop, program execution pauses to show the current value of I. When I reaches 20, execution is finally transferred out of the loop by the g ISG instruction and the program returns to line 000 and halts.

Slide the PRGM-RUN switch to PRGM and key in the following program.

Keystrokes	Display	
f CLEAR PRGM	000-	
h LBL A	001-25, 13, 11	
•	002- 73	Loop control number.
0	003- 0	
1	004- 1	
9	005- 9	
STO f I	006-23, 14, 23	Store loop control number.
h LBL 1	007-25, 13, 1	
RCL f I	008-24, 14, 23	Current integer value of I stored in storage register addressed by (i).
h INT	009- 25 32	
STO f (i)	010-23, 14, 24	
h PSE	011- 25 74	Pause to display current value of I.
g ISG	012- 15 24	Add one to value in I-register and compare with counter test value (019).
GTO 1	013- 22 1	If I≤19, execute loop again.
h RTN	014- 25 12	If I>19, execution transfers to line 000 and halts.

Slide the PRGM-RUN switch to RUN .

When the program is run, it begins by placing zero in the I-register. Then the program recalls the current value in the I-register *(loop control value)* and stores the integer part of that number in the corresponding address — for example, when the I-register contains the number 17.019, that number is recalled and the integer portion, 17, is stored in the indirect storage register (R_{17}) that is addressed by the number 17. Each time through the loop the I-register is incremented and the result is used both as data and as an address by the STO f (i) instruction. When the number in the I-register reaches 20, execution transfers out of the loop and the program stops.

To run the Program:

Keystrokes	Display
A	0.0000
	1.0000
	2.0000
	.
	.
	.
	19.0000

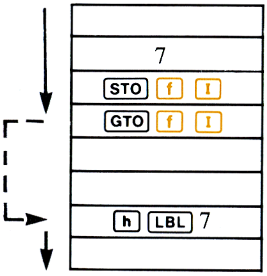
Notice that the contents of the I-register have been incremented to 20.0190.

Keystrokes	Display
RCL f I	20.0190

I-Register Control of Branches and Subroutines

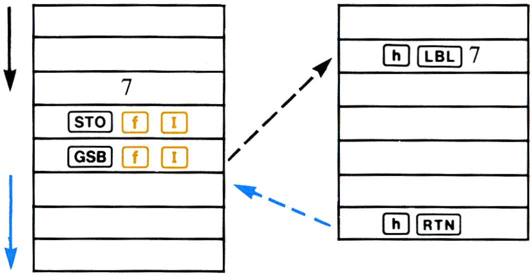
Like the addressing of storage registers using STO f (i) and RCL f (i), you can address routines, subroutines, even entire programs, with the I-register.

To address a routine using the I-register, use the instruction GTO f I. When a running program encounters a GTO f I instruction, execution is transferred sequentially downward to the LBL that is addressed by the number in the I-register. Thus, with the number 7 stored in I, when the instruction GTO f I is encountered, execution is transferred downward in program memory to the next LBL 7 instruction before resuming.



Naturally, you can also execute GTO f I from the keyboard when you want the calculator to go to the label addressed in the I-register and halt.

Subroutines can also be addressed and utilized with the I-register. When GSB f I is executed in a running program, execution transfers to the specified LBL and executes the subroutine. When a RTN is then encountered, execution transfers back to the next instruction after the GSB f I and resumes. For example, with the number 7 stored in the I-register, GSB f I causes execution of the subroutine defined by LBL 7 and RTN.



You can also execute GSB f I from the keyboard when you want the calculator to execute the program or subroutine addressed by the number in I, then halt.

The simple-to-remember addressing using the I-register is the same for GTO f I and GSB f I. If the I-register contains zero or a *positive* number from 1 through 9, GTO or GSB f I addresses LBL 0 through 9. When the number in I is a positive 10 or 11, LBL A or

$\boxed{\text{LBL}}$ $\boxed{\text{B}}$ is addressed. Label addressing is illustrated below.

If the number in I is:	$\boxed{\text{GTO}}$ $\boxed{\text{f}}$ $\boxed{\text{I}}$ or $\boxed{\text{GSB}}$ $\boxed{\text{f}}$ $\boxed{\text{I}}$ transfers execution to:
0	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 0
1	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 1
2	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 2
3	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 3
4	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 4
5	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 5
6	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 6
7	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 7
8	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 8
9	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 9
10	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{A}}$
11	$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{B}}$

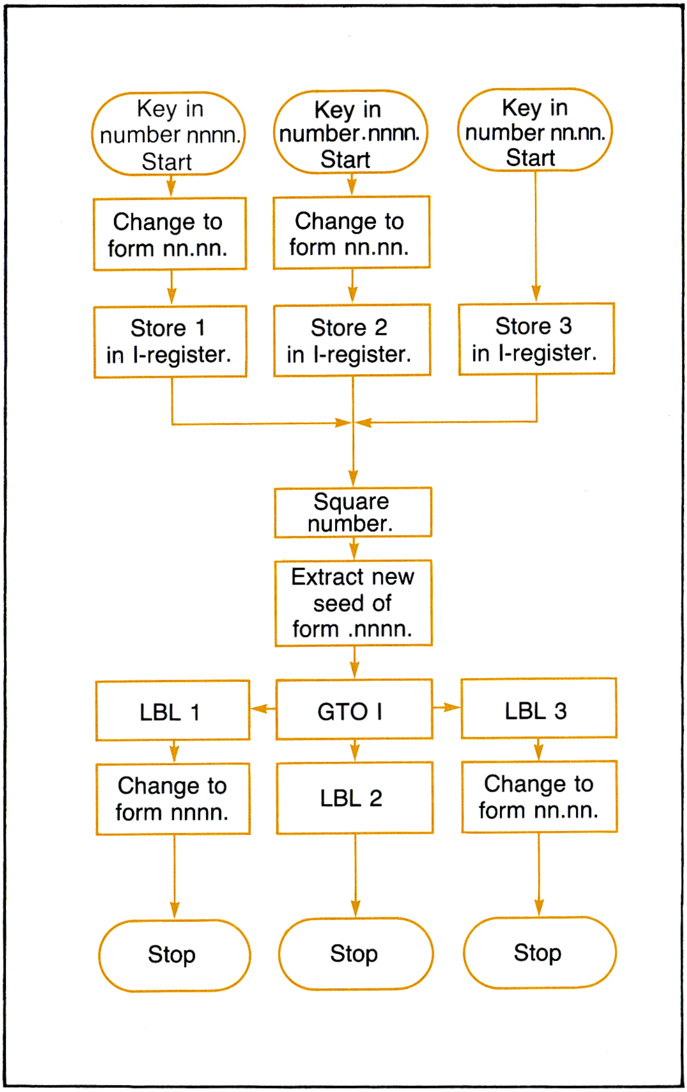
Remember that label address numbers in the I-register must be 0 or a positive value less than 12 (negative numbers cause transfer of program execution, which we will discuss later), and that the calculator looks at only the integer portion of the number in I when using it for an address.

Example: One method of generating pseudorandom numbers in a program is to take a number (called a "seed"), square it, and then remove the center of the resulting square and square that, etc. Thus, a seed of 5182 when squared yields 26853124. A random number generator could then extract the four center digits, 8531, and square that value. Continuing for several iterations through a loop would generate several pseudorandom numbers.*

The following program uses the $\boxed{\text{GTO}}$ $\boxed{\text{f}}$ $\boxed{\text{I}}$ instruction to permit you to key in a four-digit seed in any of three forms: **nnnn**, **.nnnn**, or **nn.nn**. The seed is squared and the square truncated by the main part of the program, and the resulting four-digit random number is displayed in the form of the original seed: **nnn**, **.nnnn**, or **nn.nn**.

* As indicated, the numbers are not really random. After several such "pseudorandom" numbers have been generated by this mid-square method they may well begin behaving in a very systematic, non-random way. The art of generating truly random numbers is beyond the scope of this handbook.

A flowchart for the program might look like this:



The use of the GTO f I instruction lets you select, via your seed format, the operations that are performed upon the number after the main portion of the program.

By storing 1, 2, or 3 in the I-register depending upon the format of the seed, the program selects the form of the result after it is generated by the main portion of the program. Although the program shown here stops after each result, it would be a simple matter to create a loop that would iterate several times, increasing the apparent randomness of the result each time.

Slide the PRGM-RUN switch to PRGM and key in the program.

Keystrokes	**Display**	
f CLEAR PRGM	*000–*	
h LBL 4	*001– 25, 13, 4*	
EEX	*002– 33*	
2	*003– 2*	Changes *nnnn* to *nn.nn*.
÷	*004– 71*	
1	*005– 1*	Places 1 in X-register for storage in I.
GTO 7	*006– 22 7*	
h LBL 5	*007– 25, 13, 5*	
EEX	*008– 33*	
2	*009– 2*	Changes .*nnnn* to *nn.nn*.
×	*010– 61*	
2	*011– 2*	Places 2 in X-register for storage in I.
GTO 7	*012– 22 7*	
h LBL 6	*013– 25, 13, 6*	
3	*014– 3*	Places 3 in X-register for storage in I.
h LBL 7	*015– 25, 13, 7*	
STO f I	*016– 23, 14, 23*	Stores address of later operation in I.

Keystrokes	Display	
x⇄y	017– 21	Brings *nn.nn* to X-register.
g x^2	018– 15 3	Squares *nn.nn*.
EEX	019– 33	
2	020– 2	Truncates two final digits of square.
×	021– 61	
h INT	022– 25 32	
EEX	023– 33	
4	024– 4	Truncates two leading digits of square.
÷	025– 71	
h FRAC	026– 25 33	
GTO f I	027– 22, 14, 23	Transfers execution to appropriate operational routine.
h LBL 1	028– 25, 13, 1	
EEX	029– 33	
4	030– 4	Result appears as *nnnn*.
×	031– 61	
f FIX 0	032– 14, 11, 0	
h RTN	033– 25 12	
h LBL 2	034– 25, 13, 2	
f FIX 4	035– 14, 11, 4	Result appears as .*nnnn*.
h RTN	036– 25 12	
h LBL 3	037– 25, 13, 3	
EEX	038– 33	
2	039– 2	Result appears as *nn.nn*.
×	040– 61	
f FIX 2	041– 14, 11, 2	
h RTN	042– 25 12	

We could also have stored the digits for 100 (that is, EEX 2) and recalled them for use in lines 002-003, 008-009, 019-020, and 038-039, but we have used this more straightforward program to illustrate the use of the GTO f I instruction.

When you key in a four-digit seed number in one of the three formats shown, an address (1, 2, or 3) is placed in the R_0-register. This address is used by the [GTO] [f] [I] instruction in line 27 to transfer program execution to the proper routine so that the new random number is seen in the same form as the original seed.

Now run the program for seeds of 5182, .5182 and 51.82. To run the program:

Set the calculator to RUN .

Keystrokes	Display	
5182 [GSB] 4	*8,531.*	Random number generated in the proper form.
.5182 [GSB] 5	*0.8531*	
51.82 [GSB] 6	*85.31*	

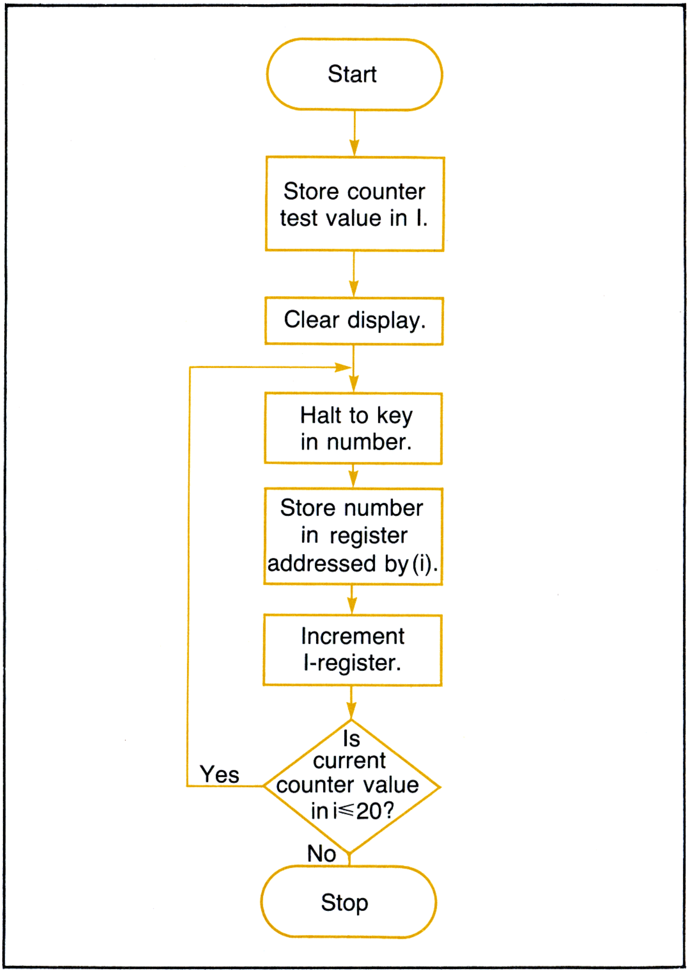
The program generates a random number of the same form as the seed you keyed in. To use the random number as a new seed (simulating the operation of an actual random number generator, in which a loop would be used to decrease the apparent predictability of each succeeding number), continue pressing [GSB] and the appropriate label key:

Keystrokes	Display
[GSB] 6	*77.79*
[GSB] 6	*51.28*
[GSB] 6	*29.63*

With a few slight modifications of the program, you could have used a [GSB] [f] [I] instruction instead of the [GTO] [f] [I] instruction.

Problem

Create and load a program using ISG and STO f (i) that permits you to key in a series of values during successive halts. The values should be stored in storage registers R_0 through R_9, $R_{.0}$ through $R_{.9}$ and I in the order you key them in. Use the following flowchart to help you.

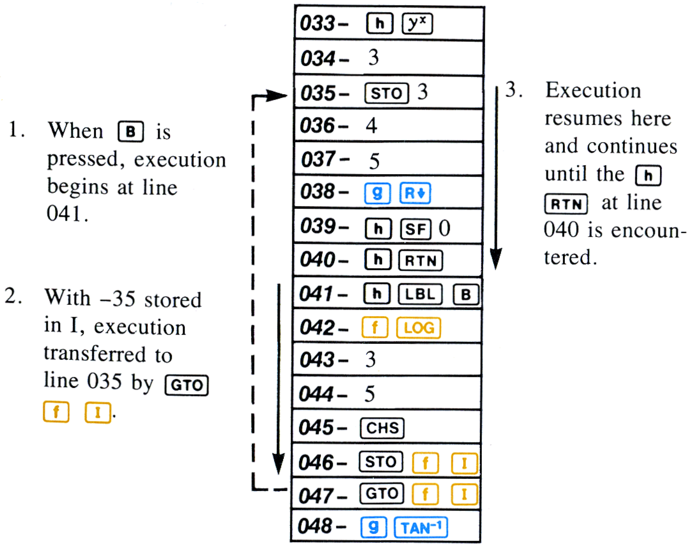

Start
Store counter test value in I.
Clear display.
Halt to key in number.
Store number in register addressed by (i).
Increment I-register.
Is current counter value in i≤20?
Yes
No
Stop

Branching and Subroutines Using Line Number Addressing

Using GTO f I or GSB f I, with a negative number stored in the I-register, you can actually branch to any occupied line number in program memory.

As you know, when GTO f I or GSB f I is executed in a running program, the calculator searches downward through program memory until it locates the LBL addressed by the positive number in I. Then execution resumes. However, when GTO f I or GSB f I is executed in a running program with a negative number stored in I, the calculator does not search for a label. Instead, execution is transferred to the occupied line number in program memory specified by the absolute value of the negative number in I. This feature allows you to transfer program execution even when all labels have been used or when you want to execute only part of a subroutine or program without using an additional label.

For example, in the section of program memory shown below, −35 is stored in the I-register. Then, when line 047, GTO f I, is executed, the running program jumps immediately to line 035, where execution begins again.



```
033- h yˣ
034- 3
035- STO 3
036- 4
037- 5
038- g R↓
039- h SF 0
040- h RTN
041- h LBL B
042- f LOG
043- 3
044- 5
045- CHS
046- STO f I
047- GTO f I
048- g TAN⁻¹
```

1. When B is pressed, execution begins at line 041.

2. With −35 stored in I, execution transferred to line 035 by GTO f I.

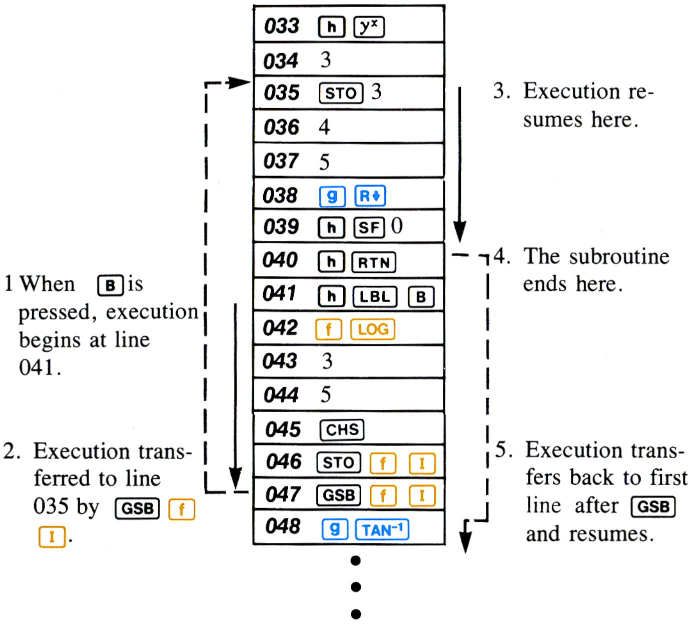
3. Execution resumes here and continues until the h RTN at line 040 is encountered.

When GTO f I is performed in a running program, execution then continues until the next RTN or R/S instruction is encountered, and then halts. If you pressed B with the instructions shown above loaded into the calculator, the instructions in lines 041 through 047 would be executed in order. Then program execution would jump backward and resume at line 035 and continue with 036, 037, etc., until the RTN instruction was encountered in line 040. Program execution would then halt and the calculator would return to line 000.

Note that executing GTO f I from the keyboard brings the same results as execution in a running program except the calculator halts at the specified line number instead of resuming program execution.

With a negative number stored in the I-register, GSB f I also transfers execution to the occupied line of program memory specified by the absolute value of the negative number in I. However, just as when using GSB with labels, subsequent instructions are then executed as a subroutine. Therefore, when the next RTN is encountered, execution transfers back to the instruction following the GSB f I instruction.

The section of program memory below shows how GSB f I operates. If you press B, −35 will be stored in the I-register. When the GSB f I at line 047 is then executed, the running program jumps back to line 035 and resumes execution. When the RTN instruction at line 040 is encountered, execution returns to line 048 and continues.



```
033 h yˣ
034 3
035 STO 3
036 4
037 5
038 g R↓
039 h SF 0
040 h RTN
041 h LBL B
042 f LOG
043 3
044 5
045 CHS
046 STO f I
047 GSB f I
048 g TAN⁻¹
•
•
•
```

1 When B is pressed, execution begins at line 041.

2. Execution transferred to line 035 by GSB f I.

3. Execution resumes here.

4. The subroutine ends here.

5. Execution transfers back to first line after GSB and resumes.

Like GTO f I, GSB f I can be used to jump to a specific line of program memory without running your entire program. When you execute GSB f I from the keyboard using the absolute value of a negative number in I as an occupied line address, the calculator jumps to that line and begins execution. However, unlike the execution of GSB f I in a running program, when a RTN is encountered, the calculator returns to line 000 and halts.

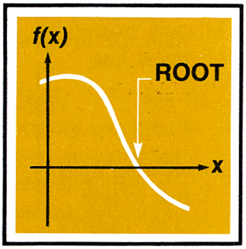
Section 8

Finding the Roots of an Equation

In many applications you need to solve equations of the form

$$f(x) = 0.^*$$

This means finding the values of x that satisfy the equation. Each such value of x is called a *root* of the equation $f(x) = 0$ and a *zero* of the function $f(x)$. These roots (or zeros) that are real numbers are called *real roots* (or real zeros). For many problems the roots of an equation can be determined analytically through algebraic manipulation; in many other instances, this is not possible. Numerical techniques can be used to estimate the roots when analytical methods are not suitable. When you use the SOLVE key on your HP-34C, you utilize an advanced numerical technique that lets you effectively and conveniently find *real roots* for a wide range of equations.



Using SOLVE

The basic rules for using SOLVE are:

1. Key in a subroutine that evaluates the function $f(x)$ that is to be equated to zero. This subroutine must begin with the instruction h LBL followed by 0, 1, 2, 3, A, or B, and must place the value of $f(x)$ into the X-register.
2. Key two initial estimates of the desired root, separated by ENTER↑, into the X- and Y-registers. These estimates merely indicate to the calculator the approximate range of x in which it should initially seek a root of $f(x) = 0$.
3. Press f SOLVE followed by the label of your subroutine. The calculator then searches for the desired zero of your function and

* Actually, *any* equation with one variable can be expressed in this form. For example, $f(x) = a$ is equivalent to $f(x) - a = 0$, and $f(x) = g(x)$ is equivalent to $f(x) - g(x) = 0$.

displays the result. If the function that you are analyzing equals zero at more than one value of x, the routine will stop when it finds any one of those values. To find additional values, you can key in different initial estimates and use SOLVE again.

Immediately before SOLVE uses your function subroutine, a value of x is placed in the X-, Y-, Z-, and T-registers. This value is then used by your subroutine to calculate $f(x)$. Because the entire stack is filled with the x value, this number is continually available to your subroutine. (The use of this technique is described on page 76).

Example: Use SOLVE to find the values of x for which

$$f(x) = x^2 - 3x - 10 = 0.$$

Using Horner's method (refer to page 79), you can rewrite $f(x)$ so that it is programmed more efficiently:

$$f(x) = (x - 3)x - 10.$$

Slide the PRGM-RUN switch to PRGM and key in the following subroutine that evaluates $f(x)$.

Keystrokes	**Display**	
f CLEAR PRGM	*000-*	Clear program memory.
h LBL 0	*001- 25, 13, 0*	Begin with LBL instruction.
3	*002- 3*	
−	*003- 41*	
×	*004- 61*	
1	*005- 1*	
0	*006- 0*	
−	*007- 41*	
h RTN	*008- 25 12*	

Now slide the PRGM-RUN switch back to PRGM ▮▮. Key two initial estimates into the X- and Y-registers. Try estimates of 0 and 10 to look for a positive root.

Keystrokes	Display*	
0 ENTER↑	*0.0000*	Initial estimates.
10	*10.*	

You can now find the desired root by pressing f SOLVE 0. When you do this, the calculator will not display the answer right away. The HP-34C uses an iterative algorithm† to estimate the root. The algorithm analyzes your function by sampling it many times, perhaps a dozen times or more. It does this by repeatedly executing your subroutine. Finding a root will usually require about 30 seconds to 2 minutes; but sometimes the process will require even more time.

Press f SOLVE 0 and sit back while your HP-34C exhibits one of its powerful capabilities:

Keystrokes	Display	
f SOLVE 0	**5.0000**	The desired root.

After the routine finds and displays the root, you can ensure that the displayed number is indeed a root of $f(x) = 0$ by checking the stack. You have seen that the displayed X-register contains the desired root. The Y-register contains a previous estimate of the root, which should be very close to the displayed root. The Z-register contains the value of your function evaluated at the displayed root.

* Press f FIX 4 to obtain the displays in this section. The display setting does not influence the operation of SOLVE.

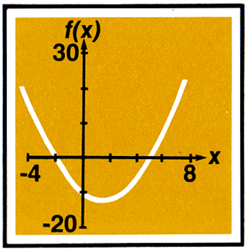
† An *algorithm* is a step-by-step procedure for solving a mathematical problem. An *iterative* algorithm is one containing a portion that is executed a number of times in the process of solving the problem.

Keystrokes	Display	
g R↓	5.0000	A previous estimate of the root.
g R↓	0.0000	Value of the function at the root, showing that $f(x) = 0$.

Quadratic equations, such as the one you are solving, can have two roots. If you specify two new initial estimates, you can check for a second root. Try estimates of 0 and −10 to look for a negative root.

Keystrokes	Display	
0 ENTER↑	0.0000	} Initial estimates.
10 CHS	−10.	
f SOLVE 0	−2.0000	The second root.
g R↓	−2.0000	A previous estimate of the root.
g R↓	0.0000	Value of $f(x)$ at second root.

You have now found the two roots of $f(x) = 0$. Note that this quadratic equation *could* have been solved algebraically—and you would have obtained the same roots that you found using SOLVE.



Graph of $f(x)$

The convenience and power of the SOLVE key becomes more apparent when you solve an equation for a root that cannot be determined algebraically.

Example: Champion ridget hurler Chuck Fahr throws a ridget with an upward velocity of 50 meters/second. If the height of the ridget is expressed as

$$h = 5000(1 - e^{-t/20}) - 200t,$$

how long does it take for it to reach the ground again? In this equation, h is the height in meters and t is the time in seconds.

Solution: The desired solution is the positive value of t at which $h = 0$.

Slide the PRGM-RUN switch to PRGM and key in the following subroutine that calculates the height.

Keystrokes	**Display**	
h LBL A	*001– 25, 13, 11*	Begin with LBL instruction.
2	*002– 2*	
0	*003– 0*	
÷	*004– 71*	
CHS	*005– 32*	
g e^x	*006– 15 1*	
CHS	*007– 32*	
1	*008– 1*	
+	*009– 51*	
5	*010– 5*	
0	*011– 0*	
0	*012– 0*	
0	*013– 0*	
×	*014– 61*	
x≷y	*015– 21*	Bring t value into X-register.

Keystrokes	Display
2	016– 2
0	017– 0
0	018– 0
×	019– 61
−	020– 41
h RTN	021– 25 12

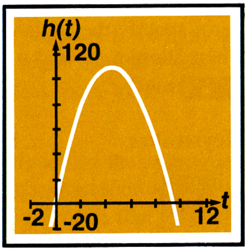
Next, set the PRGM-RUN switch to RUN. Key in two initial estimates of the time (for example, 5 and 6 seconds) and execute SOLVE.

Keystrokes	Display	
5 ENTER↑	5.0000	Initial estimates.
6	6.	
f SOLVE A	9.2843	The desired root.

Verify the root by reviewing the Y- and Z-registers.

Keystrokes	Display	
g R↓	9.2843	A previous estimate of the root.
g R↓	0.0000	Value of the function at the root, showing that $h = 0$.

Fahr's ridget falls to the ground 9.2843 seconds after he hurls it—a remarkable toss.



Graph of h versus t

When No Root Is Found

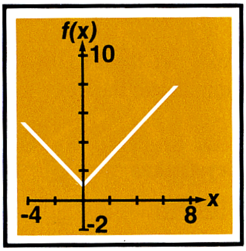
You have seen how the SOLVE key estimates and displays a root of an equation of the form $f(x) = 0$. However, it *is* possible that an equation has no real roots (that is, there is no real value of x for which the equality is true). Of course, you would not expect the HP-34C to find a root in this case. Instead, it displays ***Error 6***.

Example: Consider the equation

$$|x| = -1$$

which has no solution since the absolute value function is never negative. Express this equation in the required form

$$|x| + 1 = 0$$



Graph of $f(x) = |x| + 1$

and attempt to use SOLVE to find a solution. With the PRGM-RUN switch set to PRGM ▮, key in the required function subroutine.

Keystrokes	Display	
h LBL 1	***001– 25, 13, 1***	Begin subroutine with LBL instruction.
h ABS	***002– 25 34***	
1	***003– 1***	
+	***004– 51***	
h RTN	***005– 25 12***	

Because the absolute-value function is minimum near an argument of zero, specify the initial estimates in that region, for instance 1 and –1. Then attempt to find a root. After setting the PRGM-RUN switch to ▮ RUN :

Keystrokes	Display	
1 ENTER↑	***1.0000***	} Initial estimates.
1 CHS	***-1.***	
f SOLVE 1	***Error 6***	This display indicates that no root was found.

As you can see, the HP-34C stopped seeking a root of $f(x) = 0$ when it decided that none existed—at least not in the general range of x to which it was initially directed. The ***Error 6*** display does not indicate that an "illegal" operation has been attempted; it merely states that no root was found where SOLVE presumed one might exist (based on your initial estimates).

If the HP-34C stops seeking a root and displays an error message, one of these four types of conditions has occurred:

- If repeated iterations all produce a constant non-zero value for the specified function, execution stops with the display ***Error 6.***
- If numerous samples indicate that the *magnitude* of the function appears to have a nonzero minimum value in the area being searched, execution stops with the display ***Error 6***.
- If an improper argument is used in a mathematical operation as part of your subroutine, execution stops with the display ***Error 0***.
- If the result of any calculation has a magnitude greater than $9.999999999 \times 10^{99}$, execution stops with all 9's and the appropriate sign (or ***Error 1*** in the case of register overflow) in the display.

In the case of a constant function value, the routine can see no indication of a tendency for the value to move toward zero. This can occur for a function whose 10 most significant digits *are* constant (such as when its graph levels off at a nonzero horizontal asymptote) or for a function with a relatively broad, local "flat" region in comparison to the range of x values being tried.

In the case where the function's magnitude reaches a nonzero minimum, the routine has logically pursued a sequence of samples for which the magnitude has been getting smaller. However, it has not found a value of x at which the function's graph touches or crosses the x-axis.

The two final cases point out a potential deficiency in the subroutine rather than a limitation of the root-finding routine. Improper operations may sometimes be avoided by specifying initial estimates that focus the search in a region where such an outcome will not occur. However, the SOLVE routine is very aggressive and may sample the function over a wide range. It is a good practice to have your subroutine test or adjust potentially improper arguments prior to performing an operation (for instance, use ABS prior to $\sqrt{x}$). Rescaling variables to avoid large numbers can also be helpful.

The success of the SOLVE routine in locating a root depends primarily upon the nature of the function it is analyzing and the initial estimates at which it begins searching. The mere existence of a root does not ensure that the casual use of the SOLVE key will find it. If the function $f(x)$ has a nonzero horizontal asymptote or a local minimum of its magnitude, the routine can be expected to find a root of $f(x) = 0$ only if the initial estimates do not concentrate the search in one of these unproductive regions—and, of course, if a root actually exists.

Choosing Initial Estimates

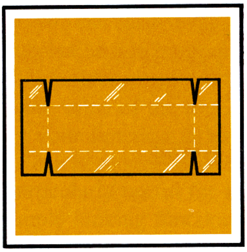
When you use SOLVE to find the root of an equation, the two initial estimates that you provide determine the values of the variable x at which the routine begins its search. In general, the likelihood that you will find the particular root you are seeking increases with the level of understanding that you have about the function you are analyzing. Realistic, intelligent estimates greatly facilitate the determination of a root.

The initial estimates that you use may be chosen in a number of ways:

If the variable x has a limited range in which it is conceptually meaningful as a solution, it is reasonable to choose initial estimates within this range. Frequently an equation that is applicable to a real problem has, in addition to the desired solution, other roots that are physically meaningless. These usually occur because the equation being analyzed is appropriate only between certain limits of the variable. You should recognize this restriction and interpret the results accordingly.

If you have some knowledge of the behavior of the function $f(x)$ as it varies with different values of x, you are in a position to specify initial estimates in the general vicinity of a zero of the function. You can also avoid the more troublesome ranges of x such as those producing a relatively constant function value or a minimum of the function's magnitude.

Example: Using a rectangular piece of sheet metal 4 decimeters by 8 decimeters, an open-top box having a volume of 7.5 cubic decimeters is to be formed. How should the metal be folded? (A tall box is preferred to a short one.)



Solution: You need to find the height of the box (that is, the amount to be folded up along each of the four sides) that gives the specified volume. If x is the height (or amount folded up), the length of the box is $(8 - 2x)$ and the width is $(4 - 2x)$. The volume V is given by

$$V = (8 - 2x)(4 - 2x)x.$$

By expanding the expression and then using Horner's method (page 79), this equation can be rewritten as

$$V = 4((x - 6)x + 8)x.$$

To get $V = 7.5$, find the values of x for which

$$f(x) = 4((x - 6)x + 8)x - 7.5 = 0.$$

Set the PRGM-RUN switch to PRGM ▮▮ and key in the following subroutine that calculates $f(x)$.

Keystrokes	**Display**	
[h] [LBL] 3	*001– 25, 13, 3*	Begin with [LBL] instruction.
6	*002– 6*	
[−]	*003– 41*	
[×]	*004– 61*	
8	*005– 8*	
[+]	*006– 51*	
[×]	*007– 61*	
4	*008– 4*	
[×]	*009– 61*	

Keystrokes	Display
7	010- 7
`.`	011- 73
5	012- 5
`-`	013- 41
h RTN	014- 25 12

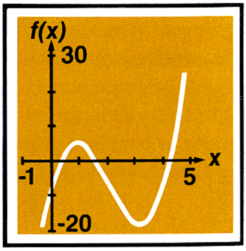
It seems reasonable that either a tall, narrow box or a short, flat box could be formed having the desired volume. Because the tall box is preferred, larger initial estimates of the height are reasonable. However, heights greater than 2 decimeters are not physically possible (because the metal is only 4 decimeters wide). Initial estimates of 1 and 2 decimeters are therefore appropriate.

Set the PRGM-RUN switch to RUN and find the desired height.

Keystrokes	Display	
1 ENTER↑	1.0000	Initial estimates.
2	2.	
f SOLVE 3	1.5000	The desired height.
g R↓	1.5000	Previous estimate.
g R↓	0.0000	$f(x)$ at root.

By making the height 1.5 decimeters, a 5.0 × 1.0 × 1.5-decimeter box is specified.

If you ignore the upper limit on the height and use initial estimates of 3 and 4 decimeters (still less than the width), you will obtain a height of 4.2026 decimeters—a root that is physically meaningless. If you use small initial estimates such as 0 and 1 decimeter, you will obtain a height of 0.2974 decimeter—producing an undesirable short, flat box.



Graph of $f(x)$

As an aid for examining the behavior of a function, you can easily evaluate the function at one or more values of x if your subroutine is in program memory. To do this, key the value of x into the X-register, then press ENTER↑ ENTER↑ ENTER↑ to fill the stack. Calculate the value of the function by pressing A, B, or GSB followed by your function label, whichever is appropriate. The values you calculate can be plotted to give you a graph of the function. This procedure is particularly useful for a function whose behavior you do not know. A simple-looking function may have a graph with relatively extreme variations that you might not anticipate. A root that occurs near a localized variation may be hard to find unless you specify initial estimates that are close to the root.

If you have no informed or intuitive concept of the nature of the function or the location of the zero you are seeking, you can search for a solution using trial-and-error. The success of finding a solution depends partially upon the function itself. Trial-and-error is often—but not always—successful.

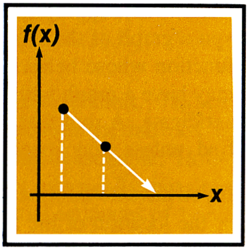
- If you specify two moderately large positive or negative estimates and the function's graph does not have a horizontal asymptote, the routine will seek a zero which might be the most positive or negative (unless the function oscillates many times, as the trigonometric functions do).
- If you have already found a zero of the function, you can check for another solution by specifying estimates that are relatively distant from any known zeros.
- Many functions exhibit special behavior when their arguments approach zero. You can check your function to determine values of x for which any argument within your function becomes zero, and then specify estimates at or near those values.

Although two different initial estimates are usually supplied when using SOLVE, you can also use SOLVE with the same estimate in both the X- and Y-registers. If the two estimates are identical, a second estimate is generated internally. If your single estimate is nonzero, the second estimate differs from your estimate by one count in the seventh significant digit. If your estimate is zero, 1×10^{-7} is used as the second estimate. Then the root-finding procedure continues as it normally would with two estimates.

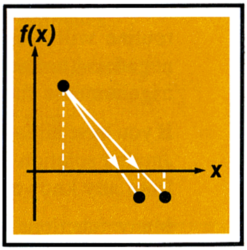
How SOLVE Works

You will be able to use SOLVE most effectively by having a basic understanding of how the algorithm works.

In the process of searching for a zero of the specified function, the algorithm uses the value of the function at two or three previous estimates to approximate the shape of the function's graph. The algorithm uses this shape to intelligently "predict" a new estimate where the graph might cross the x-axis. The function subroutine is then executed, computing the value of the function at the new estimate. This procedure is performed repeatedly by the SOLVE algorithm.



If any two estimates yield function values with opposite signs, the algorithm presumes that the function's graph must cross the x-axis in at least one place in the interval between these estimates. The interval is systematically narrowed until a root of the equation is found.



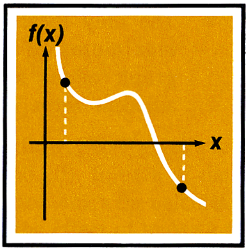
A root is successfully found either if the computed function value is equal to zero or if two estimates, differing by less than two or three units in their least-significant (tenth) digit, give function values having opposite signs. In this case, execution stops and the estimate is displayed.

As discussed earlier (refer to page 180), the occurrence of other situations in the iteration process indicate the apparent absence of a function zero. This is a result of there being no way to logically predict a new estimate that is likely to have a function value closer to zero. In such cases, ***Error 6*** is displayed.

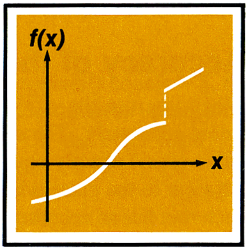
You should note that the initial estimates you provide are used to begin the "prediction" process. By permitting more accurate predictions than might otherwise occur, properly chosen estimates greatly facilitate the determination of the solution you seek.

The SOLVE algorithm will *always* find a root provided one exists, if any one of four conditions are met:

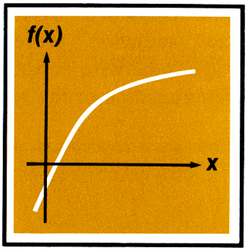
- Any two estimates have function values with opposite signs.



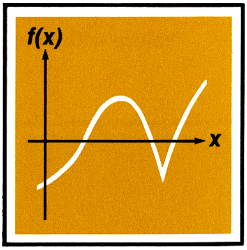
- The function is monotonic, meaning that $f(x)$ either always decreases or else always increases as x is increased.



- The function's graph is either convex everywhere or concave everywhere.



- The only local minimums and maximums of the function's graph occur singly between adjacent zeros of the function.



In addition, it is assumed that the SOLVE algorithm will not be interrupted by an improper operation or overflow condition.

Accuracy of the Root

When you use the SOLVE key to find a root of an equation, the root is found accurately. The displayed root either gives a calculated function value ($f(x)$) exactly equal to zero or else is a 10-digit number virtually adjacent to the place where the function's graph crosses the x-axis. Any such root has an accuracy within two or three units in the tenth significant digit.

In most situations the calculated root is an accurate estimate of the theoretical (infinitely precise) root of the equation. However, certain conditions can cause the finite accuracy of the calculator to give a result that appears to be inconsistent with your theoretical expectation.

If a calculation has a result whose magnitude is smaller than $1.000000000 \times 10^{-99}$, the result is set equal to zero. This effect is referred to as "underflow." If the subroutine that calculates your function encounters underflow for a range of x and if this affects the value of the function, then a root in this range may be expected to have some inaccuracy. For example, the equation

$$x^4 = 0$$

has a root at $x = 0$. Because of underflow, SOLVE produces a root of ***1.5060   -25*** (for initial estimates of 1 and 2). As another example, consider the equation

$$1/x^2 = 0$$

whose root is infinite in value. Because of underflow, SOLVE gives a root of ***3.1707   49*** (for initial estimates of 10 and 20). In each of these examples, the algorithm has found a value of x for which the calculated function value equals zero. By understanding the effect of underflow, you can readily interpret results such as these.

The accuracy of a computed value sometimes can be adversely affected by "round-off" error, by which an infinitely precise number is rounded to 10 significant digits. If your subroutine requires excessive precision to properly calculate the function for a range of x, the result obtained by SOLVE may be inaccurate. For example, the equation

$$|x^2 - 5| = 0$$

has a root at $x = \sqrt{5}$. Because no 10-digit number *exactly* equals $\sqrt{5}$, the result of using SOLVE is ***Error 6*** (for any initial estimates) because the function never equals zero nor changes sign. On the other hand, the equation

$$[(|x| + 1) + 10^{15}]^2 = 10^{30}$$

has no roots because the left side of the equation is always greater than the right side. However, because of round-off in the calculation of

$$f(x) = [(|x| + 1) + 10^{15}]^2 - 10^{30},$$

the root ***1.0000*** is found for initial estimates of 1 and 2. By recognizing situations in which round-off error may influence the operation of $\boxed{\text{SOLVE}}$, you can evaluate the results accordingly and perhaps rewrite the function to reduce the effects of round-off.

In a variety of practical applications, the parameters in an equation—or perhaps the equation itself—are merely *approximations*. Physical parameters have an inherent accuracy (or inaccuracy). Mathematical representations of physical processes are only models of those processes, accurate only to the extent that the underlying assumptions are true. An awareness of these and other inaccuracies can be used to your advantage. By structuring your subroutine to return a function value of zero when the calculated value is negligible for practical purposes, you can usually save considerable time in finding a root with $\boxed{\text{SOLVE}}$—particularly for cases that would normally take a long time.

Example: Ridget hurlers such as Chuck Fahr can throw a ridget to heights of 105 meters and more. In fact, Fahr's hurls usually reach a height of 107 meters. How long does it take for his remarkable toss, described on page 178, to reach 107 meters?

Solution: The desired solution is the value of t at which $h = 107$. The subroutine from the earlier example calculates the height of the ridget. This subroutine can be used in a new function subroutine to calculate

$$f(t) = h(t) - 107.$$

Slide the PRGM-RUN switch to PRGM ▥■ and key in a subroutine that calculates $f(t)$.

Keystrokes	**Display**	
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ $\boxed{\text{B}}$	***001– 25, 13, 12***	Begin with $\boxed{\text{LBL}}$ instruction.
$\boxed{\text{GSB}}$ $\boxed{\text{A}}$	***002– 13 11***	Calculates $h(t)$.

Keystrokes	Display	
1	*003- 1*	
0	*004- 0*	
7	*005- 7*	Calculates $h(t) - 107$.
−	*006- 41*	
h RTN	*007- 25 12*	

Now slide the PRGM-RUN switch to RUN. In order to find the first time at which the height is 107 meters, use initial estimates of 0 and 1 second.

Keystrokes	Display	
0 ENTER↑	*0.0000*	Initial estimates.
1	*1.*	
f SOLVE B	*4.1718*	The desired root.
g R↓	*4.1718*	A previous estimate of the root.
g R↓	*0.0000*	Value of $f(t)$ at root.

It takes 4.1718 seconds for the ridget to reach a height of exactly 107 meters. (It takes approximately one minute to find this solution.)

However, suppose you assume that the function $h(t)$ is accurate only to the nearest whole meter. You can now change your subroutine to give $f(t) = 0$ whenever the calculated magnitude of $f(t)$ is less than 0.5 meter. Slide the PRGM-RUN switch to PRGM and key in the following changes to your subroutine:

Keystrokes	Display	
GTO · 006	*006- 41*	Line before RTN instruction.
h ABS	*007- 25 34*	Magnitude of $f(t)$.
·	*008- 73*	Accuracy.
5	*009- 5*	
f x>y	*010- 14 51*	Return zero if accuracy > magnitude.
CLx	*011- 34*	
g x≠0	*012- 15 61*	Restore $f(t)$ if value is nonzero.
h LST x	*013- 25 0*	

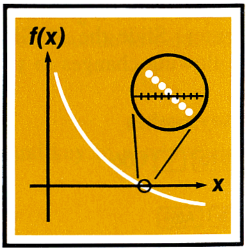
Slide the PRGM-RUN switch to RUN and execute SOLVE again.

Keystrokes	Display	
0 ENTER↑	0.0000	Initial estimates.
1	1.	
f SOLVE B	4.0681	The desired root.
g R↓	4.0681	A previous estimate of the root.
g R↓	0.0000	Value of modified $f(t)$ at root.

After 4.0681 seconds, the ridget is at a height of 107 ±0.5 meters. This solution, although different from the previous answer, is correct considering the uncertainty of the height equation. (And this solution is found in just under half the time of the earlier solution.)

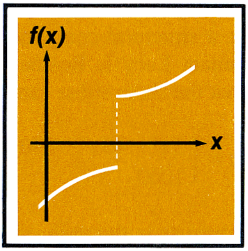
Interpreting Results

The numbers that SOLVE places in the X-, Y-, and Z-registers help you evaluate the results of the search for a root of your equation.* Even when no root is found, the results are still significant.

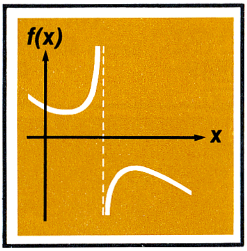


When SOLVE finds a root of the specified equation, the root and function values are placed in the X- and Z-registers. A function value of zero is the expected result. However, a nonzero function value is also acceptable because it indicates that the function's graph apparently crosses the x-axis within an infinitesimal distance from the calculated root. In most such cases, the function value will be relatively close to zero.

* The number in the T-register is the same number that was left in the Y-register by the final execution of your function subroutine. Generally, this number is not of interest.



Special consideration is required for a different type of situation in which SOLVE finds a root with a nonzero function value. If your function's graph has a discontinuity that crosses the *x*-axis, SOLVE specifies as a root an *x* value adjacent to the discontinuity. This is reasonable because a large change in the function value between two adjacent values of *x* might be the result of a very rapid, continuous transition. Because this cannot be resolved by the algorithm, the root is displayed for you to interpret.



A function may have a *pole,* where its value approaches infinity. If the function value changes sign at a pole, the corresponding value of *x* looks like a possible root of your equation, just as it would for any other discontinuity crossing the *x*-axis. However, for such functions, the function value placed into the Z-register when that root is found will be relatively large. If the pole occurs at a value of *x* that is *exactly* represented with 10 digits, the subroutine may try that value and halt prematurely with an error or overflow indication. In this case, the SOLVE operation will not be completed. Of course, this may be avoided by the prudent use of a conditional statement in your subroutine.

Example: In her analysis of the stresses in a structural component, design consultant Lucy I. Beame has determined that the shear stress can be expressed as

$$Q = \begin{cases} 3x^3 - 45x^2 + 350 & \text{for } 0<x<10 \\ 1000 & \text{for } 10\leqslant x<14 \end{cases}$$

where Q is the shear stress in newtons and x is the distance from one end in meters. Write a subroutine to compute the shear stress for any value of x. Use SOLVE to find the location of zero shear stress.

Solution: The equation for the shear stress for x between 0 and 10 is more efficiently programmed after rewriting it using Horner's method:

$$Q = (3x - 45)x^2 + 350 \qquad \text{for } 0<x<10.$$

Slide the PRGM-RUN switch to PRGM and key in the subroutine:

Keystrokes	**Display**	
f CLEAR PRGM	*000–*	Clear program memory.
h LBL 2	*001– 25, 13, 2*	Begin with LBL instruction.
1	*002– 1*	Test for x range.
0	*003– 0*	
f x≤y	*004– 14 41*	
GTO 9	*005– 22 9*	Branch for $x\geqslant 10$.
CLX	*006– 34*	
3	*007– 3*	
×	*008– 61*	
4	*009– 4*	
5	*010– 5*	
−	*011– 41*	
×	*012– 61*	
×	*013– 61*	
3	*014– 3*	
5	*015– 5*	
0	*016– 0*	

Keystrokes	Display
+	017- 51
h RTN	018- 25 12
h LBL 9	019- 25, 13, 9
EEX	020- 33
3	021- 3
h RTN	022- 25 12

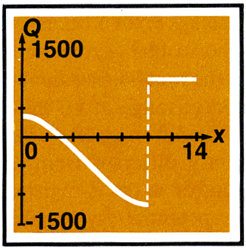
Now slide the PRGM-RUN switch to RUN. Use initial estimates of 7 and 14 to start at the outer end of the beam and search for a point of zero shear stress.

Keystrokes	Display	
7 ENTER↑	7.0000	Initial estimates.
14	14.	
f SOLVE 2	10.0000	Possible root.
g R↓ g R↓	1,000.0000	Stress not zero.

The large stress value at the root points out that the SOLVE routine has found a discontinuity. This is a place on the beam where the stress quickly changes from negative to positive. Start at the other end of the beam (estimates of 0 and 7) and use SOLVE again.

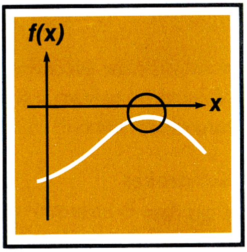
Keystrokes	Display	
0 ENTER↑	0.0000	Initial estimates.
7	7.	
f SOLVE 2	3.1358	Possible root.
g R↓ g R↓	2.0000 -07	Negligible stress.

Beame's beam has zero shear stress at approximately 3.1358 meters and an abrupt change of stress at 10.0000 meters.



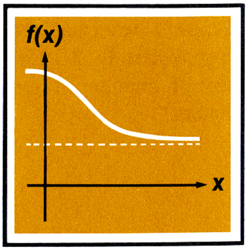
Graph of Q versus x

When no root is found and **Error 6** is displayed, you can press any key to clear the display and observe the estimate at which the function was closest to zero. By also reviewing the numbers in the Y- and Z-registers, you can often determine the nature of the function near the root estimate and use this information constructively.



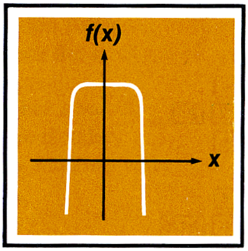
If the algorithm terminates its search near a local minimum of the function's *magnitude,* clear the **Error 6** display and observe the numbers in the X-, Y-, and Z-registers by rolling down the stack. If the value of the function saved in the Z-register is relatively close to zero, it is possible that a root of your equation has been found—the number returned in the X-register may be a 10-digit number very close to a theoretical root. You can explore this potential minimum further by rolling the stack until the returned estimates are back in the X- and Y-registers and then executing SOLVE again using these numbers as initial estimates. If an actual minimum has been found, **Error 6** will again be displayed and the number in the X-register will be approximately the same as before, but possibly closer to the actual location of the minimum.

Of course, you may deliberately use SOLVE to find the location of a local minimum of the function's magnitude. However, in this case you must be careful to confine the search in the region of the minimum. Remember, SOLVE tries hard to find a *zero* of the function.



If the algorithm stops searching and displays **Error 6** because it is working on a horizontal asymptote (when the value of the function is essentially constant for a large range of x), the estimates in the X- and Y-registers usually are significantly different from each other. The number in the Z-register is the value of the potential asymptote. If you execute SOLVE again using as initial estimates the numbers that were returned in the X- and Y-registers, a horizontal asymptote may again cause ***Error 6***, but with numbers in the X- and Y-registers that will differ from the previous numbers. The value of the function in the Z-register would then be the same as that obtained previously.

If ***Error 6*** is displayed as a result of a search that is concentrated in a local "flat" region of the function, the estimates in the X- and Y-registers will be relatively close together or extremely small. Execute SOLVE again using for initial estimates the numbers from the X- and Y-registers (or perhaps two numbers somewhat further apart). If the magnitude of the function is not a minimum nor constant, the algorithm will eventually expand its search and find a more significant result.



Example: Investigate the behavior of the function

$$f(x) = 3 + e^{-|x|/10} - 2e^{x^2e^{-|x|}} .$$

First set the PRGM-RUN switch to PRGM ▮ and key in the following subroutine to calculate $f(x)$.

Keystrokes	**Display**	
h LBL 0	***001– 25, 13, 0***	Begin with LBL instruction.
h ABS	***002– 25 34***	
CHS	***003– 32***	
g e^x	***004– 15 1***	
$x⇄y$	***005– 21***	Bring x value into X-register.
g x^2	***006– 15 3***	
×	***007– 61***	
g e^x	***008– 15 1***	
2	***009– 2***	
×	***010– 61***	
CHS	***011– 32***	

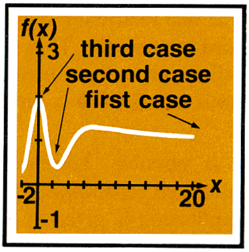
Keystrokes	Display	
x≷y	*012– 21*	Bring x value into X-register.
h ABS	*013– 25 34*	
CHS	*014– 32*	
1	*015– 1*	
0	*016– 0*	
÷	*017– 71*	
g eˣ	*018– 15 1*	
+	*019– 51*	
3	*020– 3*	
+	*021– 51*	
h RTN	*022– 25 12*	

Slide the PRGM-RUN switch to RUN and use SOLVE with the following *single* initial estimates: 10, 1, and 10^{-20}.

Keystrokes	Display	
10 ENTER↑	*10.0000*	Single estimate.
f SOLVE 0	*Error 6*	
CLx	*455.4335*	Best x value.
g R↓	*48,026,721.85*	Previous value.
g R↓	*1.0000*	Function value.
f R↑ f R↑	*455.4335*	Restore the stack.
f SOLVE 0	*Error 6*	
CLx	*48,026,721.85*	Another x value.
g R↓ g R↓	*1.0000*	Same function value (an asymptote).
1 ENTER↑	*1.0000*	Single estimate.
f SOLVE 0	*Error 6*	
CLx	*2.1213*	Best x value.
g R↓	*2.1471*	Previous value.
g R↓	*0.3788*	Function value.
f R↑ f R↑	*2.1213*	Restore the stack.
f SOLVE 0	*Error 6*	
CLx	*2.1213*	Same x value.
g R↓ g R↓	*0.3788*	Same function value (a minimum).

Keystrokes	Display	
EEX CHS 20 ENTER↑	*1.0000 -20*	Single Estimate.
f SOLVE 0	*Error 6*	
CLx	*1.0000 -20*	Best x value.
g R↓	*1.1250 -20*	Previous value.
g R↓	*2.0000*	Function value.
f R↑ f R↑	*1.0000 -20*	Restore the stack.
f SOLVE 0	*Error 6*	
CLx	*1.1250 -20*	Another x value.
g R↓	*1.5626 -16*	Previous value.
g R↓	*2.0000*	Same function value.

In each of the three cases, SOLVE initially searched for a root in a direction suggested by the graph around the initial estimate. Using 10 as the initial estimate, SOLVE found the horizontal asymptote (value of 1.0000). Using 1 as the initial estimate, a minimum of 0.3788 at $x = 2.1213$ was found. Using 10^{-20} as the initial estimate, the function was essentially constant (at a value of 2.0000) for the small range of x that was sampled.



Using SOLVE in a Program

You can use the SOLVE operation as part of a program. Be sure that the program provides initial estimates in the X- and Y-registers just prior to the SOLVE operation. The SOLVE routine stops with a value of x in the X-register and the corresponding function value in the Z-register. If the x value is a root (as explained on page 192), the program proceeds to the next line. If the x value isn't a root (as explained on page 196), the next line is skipped. Essentially, the SOLVE instruction tests whether the x value is a root and then proceeds according to the "DO IF TRUE" rule. The program can then handle the case of not finding a root, such as by choosing new initial estimates or changing a function parameter.

The use of SOLVE as an instruction in a program utilizes one of the six pending returns in the calculator. Since the subroutine called by SOLVE utilizes another return, there can be only four other pending returns. Executed from the keyboard, on the other hand, SOLVE itself does not utilize one of the pending returns, so that five pending returns are available for subroutines within the subroutine called by SOLVE. Remember that if all six pending returns have been utilized, a call to another subroutine will result in a display of ***Error 8.*** (Refer to page 135).

Restriction on the Use of SOLVE

The one restriction regarding the use of SOLVE is that SOLVE cannot be used recursively. That is, you cannot use SOLVE in a subroutine that is called during the execution of SOLVE. If this situation occurs, execution stops and ***Error 5*** is displayed.

It *is* possible, however, to use SOLVE with $\int_y^x$, thereby using the advanced capabilities of both of these keys. An example of a combined application is given in appendix A.

For Further Information

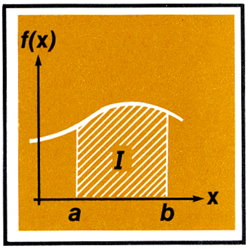
In appendix A, Advanced Use of SOLVE, additional techniques and applications for using SOLVE are presented. These include:

- Using SOLVE with polynomials.
- Finding several roots.
- Finding local extremes of a function.
- Limiting the estimation time.
- Using SOLVE with $\int_y^x$.

Section 9

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, 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 $f(x)$, the x-axis, and the limits $x = a$ and $x = b$.*

When an integral is difficult or impossible to evaluate by analytical methods, it can be calculated using numerical techniques. In the past, this could be done only with a fairly complicated computer program. With your HP-34C, however, you can easily do numerical integration using the $\boxed{\int_y^x}$ *(integrate)* key.

Using $\boxed{\int_y^x}$

The basic rules for using $\boxed{\int_y^x}$ are:

1. Key in a subroutine that evaluates the function $f(x)$ that you want to integrate. This subroutine must begin with the instruction $\boxed{h}$ $\boxed{\text{LBL}}$ followed by 0, 1, 2, 3, $\boxed{A}$, or $\boxed{B}$, and must place the value of $f(x)$ in the X-register.
2. Key the lower limit of integration (a) into the displayed X-register, then press $\boxed{\text{ENTER}\uparrow}$ to lift it into the Y-register.
3. Key the upper limit of integration (b) into the X-register.
4. Press $\boxed{f}$ $\boxed{\int_y^x}$ followed by the label of your subroutine.

* Provided that $f(x)$ is nonnegative throughout the interval of integration.

Example: Certain problems in physics and engineering require calculating *Bessel functions*. The Bessel function of the first kind of order 0 can be expressed as

$$J_0(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin\theta)\, d\theta.$$

Find $$J_0(1) = \frac{1}{\pi} \int_0^{\pi} \cos(\sin\theta)\, d\theta.$$

First, slide the PRGM-RUN switch to PRGM ▮▮ and key in the following subroutine that evaluates the function $f(\theta) = \cos(\sin\theta)$.

Keystrokes	Display	
f CLEAR PRGM	000–	Clear program memory.
h LBL 0	001– 25, 13, 0	Begin subroutine with a LBL instruction. Subroutine assumes a value of θ is in X-register.
f SIN	002– 14 7	Calculate $\sin\theta$.
f COS	003– 14 8	Calculate $\cos(\sin\theta)$.
h RTN	004– 25 12	

Now, slide the PRGM-RUN switch back to ▮▮ RUN, and key the lower limit of integration into the Y-register and the upper limit into the X-register. For this particular problem, you also need to specify radians mode for the trigonometric functions.

Keystrokes	Display	
0 ENTER↑	0.0000	Key lower limit, 0, into Y-register.
h π	3.1416	Key upper limit, π, into X-register.
g RAD	3.1416	Specify radians mode for trigonometric functions.

Now you are ready to press $\boxed{f}$ $\boxed{\int_y^x}$ 0 to calculate the integral. When you do so, you'll find that—just as with $\boxed{\text{SOLVE}}$—the calculator will not display the result right away, as it does with other operations. Your HP-34C calculates integrals using a sophisticated iterative algorithm. Briefly, this algorithm evaluates $f(x)$, the function to be integrated, at many values of x between the limits of integration. At each of these values, the calculator evaluates the function by executing the subroutine you write for that purpose. You may recall that some of the programs and subroutines you executed earlier in this handbook required several seconds to yield an answer. This may not seem too long, but when the calculator must execute the subroutine many times—as it does when you press $\boxed{\int_y^x}$—you can't expect an answer right away. Most integrals will require on the order of 30 seconds to 2 minutes; but some integrals will require even more. Later on we'll discuss how you can decrease the time somewhat; but for now, press $\boxed{f}$ $\boxed{\int_y^x}$ 0 and take a break (or read ahead) while your HP-34C takes care of the drudgery for you.

Keystrokes	**Display**	
$\boxed{f}$ $\boxed{\int_y^x}$ 0	***2.4040***	$= \int_0^\pi \cos(\sin\theta)\, d\theta.$

In general, don't forget to multiply the value of the integral by whatever constants, if any, are outside the integral. In this particular problem, we need to multiply the integral by $1/\pi$ to get $J_0(1)$:

Keystrokes	**Display**	
$\boxed{h}$ $\boxed{\pi}$	***3.1416***	
$\boxed{\div}$	***0.7652***	$J_0(1)$.

Before calling the subroutine that evaluates $f(x)$, the $\boxed{\int_y^x}$ algorithm—just like the $\boxed{\text{SOLVE}}$ algorithm—places the value of x in the X-, Y-, Z-, and T-registers. Because every stack register contains the x value, your subroutine can calculate with this number without having to recall it from a storage register. The subroutines in the next two examples take advantage of this feature. (A polynomial evaluation technique that assumes the stack is filled with the value of x is discussed on page 79.)

Note: Since the calculator puts the value of *x* into all the stack registers, any numbers previously there will be replaced by *x*. Therefore, if the stack contains intermediate results that you'll need after you calculate an integral, store those numbers in storage registers and recall them later.

Occasionally you may want to use the subroutine that you wrote for the $\int_y^x$ operation to merely evaluate the function at some value of *x*. If you do so with a function that gets *x* from the stack more than once, be sure to fill the stack manually with the value of *x*, by pressing ENTER↑ ENTER↑ ENTER↑, before you execute the subroutine.

Example: The Bessel function of the first kind of order 1 can be expressed as

$$J_1(x) = \frac{1}{\pi}\int_0^{\pi} \cos(\theta - x \sin\theta)\,d\theta.$$

Find

$$J_1(1) = \frac{1}{\pi}\int_0^{\pi} \cos(\theta - \sin\theta)\,d\theta.$$

First, slide the PRGM-RUN switch to PRGM and key in the following subroutine that evaluates the function $f(\theta) = \cos(\theta - \sin\theta)$.

Keystrokes	Display	
h LBL 1	*001– 25, 13, 1*	Begin subroutine with a LBL instruction.
f SIN	*002– 14 7*	Calculate sin θ.
−	*003– 41*	Since a value of θ will be placed into the Y-register by the $\int_y^x$ algorithm before it executes this subroutine, the − operation at this point will calculate $(\theta - \sin\theta)$.
f COS	*004– 14 8*	Calculate cos $(\theta - \sin\theta)$.
h RTN	*005– 25 12*	

Now, slide the PRGM-RUN switch back to RUN, and key the limits of integration into the X- and Y-registers. Ensure that the trigonometric mode is set to radians, then press f ∫ₓʸ 1 to calculate the integral. Finally, multiply the integral by $1/\pi$ to calculate $J_1(1)$.

Keystrokes	Display	
0 ENTER↑	**0.0000**	Key lower limit into Y-register.
h π	**3.1416**	Key upper limit into X-register.
g RAD	**3.1416**	Ensure that trigonometric mode is set to radians. (This step is not necessary if you have not switched your calculator off nor reset the trigonometric mode since you last set it to radians.)
f ∫ₓʸ 1	**1.3825**	$= \int_0^\pi \cos(\theta - \sin\theta)\, d\theta$.
h π ÷	**0.4401**	$J_1(1)$.

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

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

Find $Si(2)$.

First, slide the PRGM-RUN switch to PRGM ▮ and key in the following subroutine that evaluates the function $f(x) = (\sin x)/x$.*

Keystrokes	Display	
h LBL 2	001– 25, 13, 2	Begin subroutine with a LBL instruction.
f SIN	002– 14 7	Calculate sin x.
x⇄y	003– 21	Since a value of x will be placed in the Y-register by the ∫ algorithm before it executes this subroutine, the x⇄y operation at this point will return x to the X-register and move sin x to the Y-register.
÷	004– 71	Divide sin x by x.
h RTN	005– 25 12	

Now, slide the PRGM-RUN switch back to ▮ RUN, and key the limits of integration into the X- and Y-registers. Ensure that the trigonometric mode is set to radians, then press f ∫ 2 to calculate the integral.

* If the calculator attempted to evaluate $f(x) = (\sin x)/x$ at $x = 0$, the lower limit of integration, it would terminate with ***Error 0*** in the display (signifying an attempt to divide by zero), and the integral could not be calculated. However, the ∫ algorithm normally does *not* evaluate functions at either limit of integration, so the calculator *can* calculate the integral of a function that is undefined there. Only when the endpoints of the interval of integration are extremely close together, or the number of sample points is extremely large, does the algorithm evaluate the function at the limits of integration.

Keystrokes	Display	
0 $\boxed{\text{ENTER}\uparrow}$	**0.0000**	Key lower limit into Y-register.
2	**2.**	Key upper limit into X-register.
$\boxed{g}$ $\boxed{\text{RAD}}$	**2.0000**	Ensure that trigonometric mode is set to radians. (This step is not necessary if you have not switched your calculator off nor reset the trigonometric mode since you last set it to radians.)
$\boxed{f}$ $\boxed{\int_y^x}$ 2	**1.6054**	*Si*(2).

Accuracy of $\boxed{\int_y^x}$

The accuracy of the integral of any function depends on the accuracy of the function itself. Therefore, the accuracy of an integral calculated using $\boxed{\int_y^x}$ is limited by the accuracy of the function calculated by your subroutine.* To specify the accuracy of the function, set the display format so that the display shows *no more* than the number of digits that you consider accurate in the function's values.† If you specify fewer digits, the calculator will compute the integral more quickly;‡ but it will presume that the function is accurate to only the number of digits specified in the display format. We'll show you how you can determine the accuracy of the calculated integral after we say another word about the display format.

* It is possible that integrals of functions with certain characteristics (such as spikes or very rapid oscillations) *might* be calculated inaccurately. However, *this possibility is very small*. The general characteristics of functions that could cause problems, as well as techniques for dealing with them, are discussed in appendix B.

† The accuracy of a calculated function depends on such considerations as the accuracy of empirical constants in the function as well as round-off error in the calculations. These considerations are discussed in more detail in appendix B.

‡ The reason for this is discussed in appendix B.

You'll recall that your HP-34C provides three types of display formatting: FIX, SCI, and ENG. Which display format should be used is largely a matter of convenience, since for many integrals you'll get about the same results using any of them (provided that the number of digits is specified correctly, considering the magnitude of the function). Because it's more convenient to use SCI display format when calculating most integrals, we'll use SCI when calculating integrals in examples throughout the rest of this handbook.

Note: Remember that once you have set the display format to SCI, ENG, or FIX, you can change the number of digits appearing in the display by storing a number in the I-register and then pressing h DSP I, as described in section 7. This capability is especially useful when $\int_y^x$ is executed as part of a program, and is essential in a particular situation described in appendix B under Calculating Integrals of Maximum Accuracy.

Because the accuracy of any integral is limited by the accuracy of the function (as indicated in the display format), the calculator cannot compute the value of an integral exactly, but rather only *approximates* it. Your HP-34C places the uncertainty* of an integral's approximation in the Y-register at the same time it places the approximation in the X-register. To determine the accuracy of an approximation, check its uncertainty by pressing $x \rightleftarrows y$.

* No algorithm for numerical integration can compute the exact difference between its approximation and the actual integral. But the algorithm in your HP-34C computes an "upper bound" on this difference, which is the *uncertainty* of the approximation. For example, if the integral $Si(2)$ is 1.6054 ± 0.0001, the approximation to the integral is 1.6054 and its uncertainty is 0.0001. This means that while we don't know the exact difference between the actual integral and its approximation, we *do* know that the difference is no bigger than 0.0001.

Example: With the display format set to $\boxed{\text{SCI}}$ 2, calculate the integral in the expression for $J_1(1)$ (from the example on page 205).

Keystrokes	Display		
0 $\boxed{\text{ENTER}\uparrow}$	*0.0000*		Key lower limit into Y-register.
$\boxed{\text{h}}$ $\boxed{\pi}$	*3.1416*		Key upper limit into X-register.
$\boxed{\text{g}}$ $\boxed{\text{RAD}}$	*3.1416*		Ensure that trigonometric mode is set to radians. (This step is not necessary if you have not switched your calculator off nor reset the trigonometric mode since you last set it to radians.)
$\boxed{\text{f}}$ $\boxed{\text{SCI}}$ 2	*3.14*	*00*	Set display format to $\boxed{\text{SCI}}$ 2.
$\boxed{\text{f}}$ $\boxed{\int_y^x}$ 1	*1.38*	*00*	Integral approximated in $\boxed{\text{SCI}}$ 2.
$\boxed{x\gtrless y}$	*1.88*	*-03*	Uncertainty of $\boxed{\text{SCI}}$ 2 approximation.

The integral is 1.38 ± 0.00188. 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. In general, though, it is difficult to anticipate how many digits in an approximation will be unaffected by its uncertainty. This depends on the particular function being integrated, the limits of integration, and the display format.

If the uncertainty of an approximation is larger than what you choose to tolerate, you can decrease it by specifying a greater number of digits in the display format and repeating the approximation.*

* Provided that $f(x)$ is still calculated accurately to the number of digits shown in the display.

Whenever you want to repeat an approximation, your HP-34C can save you the trouble of keying the limits of integration back into the X- and Y-registers. After an integral is calculated, not only are the approximation and its uncertainty placed in the X- and Y-registers, but in addition the upper limit of integration is placed in the Z-register, and the lower limit is placed in the T-register. To return the limits to the X- and Y-registers for calculating an integral again, simply press [g] [R↓] [g] [R↓].

Example: For the integral in the expression for $J_1(1)$, you want an answer accurate to four decimal places instead of only two.

Keystrokes	Display	
[f] [SCI] 4	*1.8826 -03*	Set display format to [SCI] 4.
[g] [R↓] [g] [R↓]	*3.1416 00*	Roll down stack until upper limit appears in X-register.
[f] [∫ₓʸ] 1	*1.3825 00*	Integral approximated in [SCI] 4.
[x⇄y]	*1.7091 -05*	Uncertainty of [SCI] 4 approximation.

The uncertainty indicates that this approximation is accurate to at least four decimal places. Note that the uncertainty of the [SCI] 4 approximation is about one-hundredth as large as the uncertainty of the [SCI] 2 approximation. In general, the uncertainty of any [∫ₓʸ] approximation decreases by about a factor of 10 for each additional digit specified in the display format.

In the preceding example, the uncertainty indicated that the approximation *might* be correct to only four decimal places. If we temporarily display all 10 digits of the approximation, however, and compare it to the actual value of the integral (actually, an approximation known to be accurate to a sufficient number of decimal places), we find that the approximation is actually more accurate than its uncertainty indicates.

Keystrokes	Display	
[x⇄y]	*1.3825 00*	Return approximation to display.
[h] [MANT]	*1382459676*	All 10 digits of approximation.

The value of this integral, correct to eight decimal places, is 1.38245969. The calculator's approximation is accurate to *seven* decimal places rather than only four. In fact, since the uncertainty of an approximation is calculated very conservatively, *the calculator's approximation in most cases will be more accurate than its uncertainty indicates.* However, normally there is no way to determine just how accurate an approximation is; we know only that the difference between it and the actual integral is no bigger than the number in the Y-register.

We'll take a more detailed look at the accuracy and uncertainty of $\int_y^x$ approximations in appendix B.

Using $\int_y^x$ in a Program

$\int_y^x$ can appear as an instruction in a program provided that the program is not called (as a subroutine) by $\int_y^x$ itself. In other words, $\int_y^x$ cannot be used recursively. Consequently, you cannot use $\int_y^x$ to calculate multiple integrals; if you attempt to do so, the calculator will halt with ***Error 5*** in the display. However, $\int_y^x$ *can* appear as an instruction in a subroutine called by SOLVE . An example of doing so will be shown at the end of appendix A.

The use of $\int_y^x$ as an instruction in a program utilizes one of the six pending returns in the calculator. Since the subroutine called by $\int_y^x$ utilizes another return, there can be only four other pending returns. Executed from the keyboard, on the other hand, $\int_y^x$ itself does not utilize one of the pending returns, so that five pending returns are available for subroutines within the subroutine called by $\int_y^x$. Remember that if all six pending returns have been utilized, a call to another subroutine will result in a display of ***Error 8.*** (Refer to page 135).

For Further Information

This section has given you the information you need to use $\boxed{\int_y^x}$ with confidence over a wide range of applications. In appendix B, A More Detailed Look at $\boxed{\int_y^x}$, we will discuss more esoteric aspects of $\boxed{\int_y^x}$. These include:

- How $\boxed{\int_y^x}$ works.
- Accuracy, uncertainty, and calculation time.
- Accuracy of the function to be integrated.
- Uncertainty and the display format.
- Calculating integrals of maximum accuracy.
- Obtaining the current approximation to an integral.
- Considerations that could cause incorrect results.
- Considerations that prolong calculation time.

Appendix A

Advanced Use of SOLVE

Section 8 includes the basic information needed for the effective use of the SOLVE algorithm. This appendix presents more advanced, supplemental considerations regarding SOLVE.

Using SOLVE With Polynomials

In many practical applications, functions known as *polynomials* are useful for representing physical processes or more complex mathematical functions. Polynomials are easily understood and can be structured to have a wide range of mathematical characteristics.

A polynomial of degree n can be represented as

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0.$$

This function has at most n real values for which the function equals zero. A limit to the number of *positive* zeros of this function can be determined by counting the number of times the signs of the coefficients change as you scan the polynomial from left to right. Similarly, a limit to the number of *negative* zeros can be determined by scanning a new function obtained by substituting $-x$ in place of x in the original polynomial. If the actual number of real positive or negative zeros is less than its limit, it will differ by an even number. (These relationships are known as Descartes' Rule of Signs.)

As an example, consider the third-degree polynomial function

$$f(x) = x^3 - 3x^2 - 6x + 8.$$

It can have no more than three real zeros. It has at most two positive real zeros (observe the sign changes from the first to second and third to fourth terms) and one negative real zero (obtained from $f(-x) = -x^3 -3x^2 + 6x + 8$).

Polynomial functions are best programmed by rewriting them in a slightly different form that uses nested multiplication. This is sometimes referred to as Horner's method. As an illustration, the function from the previous example can be rewritten as

$$f(x) = \left[(x - 3)x - 6\right] x + 8.$$

This representation is more easily programmed and more efficiently executed than the original form, especially since the stack contains the value of x in all four registers. (This technique is described on page 79.)

Example: During the winter of '78, Arctic explorer Jean-Claude Coulerre, isolated at his frozen camp in the far north, began scanning the southern horizon in anticipation of the sun's reappearance. Coulerre knew that the sun would not be visible to him until early March, when it reached a declination of 5° 18′S. On what day and time in March was the chilly explorer's vigil rewarded?

Solution: The time in March when the sun reached 5° 18′S declination can be computed by solving the following equation for t:

$$D = a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0$$

where D is the declination in degrees, t is the time in days, and

$$\begin{aligned} a_4 &= 4.2725 \times 10^{-8} \\ a_3 &= -1.9931 \times 10^{-5} \\ a_2 &= 1.0229 \times 10^{-3} \\ a_1 &= 3.7680 \times 10^{-1} \\ a_0 &= -8.1806. \end{aligned}$$

This equation is valid for $1 \leq t < 32$, representing March, 1978.

First convert 5° 18′S to decimal degrees by pressing 5.18 CHS g →H and obtaining ***-5.3000*** (using FIX 4 display mode). (Southern latitudes are expressed as negative numbers for calculation purposes.)

The solution to Coulerre's problem is the value of t satisfying

$$-5.3000 = a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0.$$

Expressed in the form required by SOLVE, the equation is

$$0 = a_4t^4 + a_3t^3 + a_2t^2 + a_1t - 2.8806$$

where the last, constant term now incorporates the value of the declination.

Using Horner's method, the function to be set equal to zero is

$$f(t) = (((a_4t + a_3)t + a_2)t + a_1)t - 2.8806.$$

To shorten the subroutine, store and recall the constants using the registers corresponding to the exponent of t. Slide the PRGM-RUN switch to PRGM and key in the subroutine:

Keystrokes	Display	
f CLEAR PRGM	*000-*	Clear program memory.
h LBL A	*001-25, 13, 11*	Begins with LBL instruction.
RCL 4	*002- 24 4*	
×	*003- 61*	
RCL 3	*004- 24 3*	
+	*005- 51*	
×	*006- 61*	
RCL 2	*007- 24 2*	
+	*008- 51*	
×	*009- 61*	
RCL 1	*010- 24 1*	
+	*011- 51*	
×	*012- 61*	
RCL 0	*013- 24 0*	
+	*014- 51*	
h RTN	*015- 25 12*	

Now set the PRGM-RUN switch to RUN and key in the five coefficients:

Keystrokes	Display *		
4.2725 EEX CHS 8	4.2725	-08	
STO 4	4.2725	-08	Coefficient of t^4.
1.9931 CHS EEX			
CHS 5 STO 3	-1.9931	-05	Coefficient of t^3.
1.0229 EEX CHS 3	1.0229	-03	
STO 2	0.0010		Coefficient of t^2.
3.7680 EEX CHS 1	3.7680	-01	
STO 1	0.3768		Coefficient of t.
2.8806 CHS STO 0	-2.8806		Constant term.

Because the desired solution should be between 1 and 32, key in these two values for initial estimates. Then use SOLVE to find the root.

Keystrokes	Display	
1 ENTER↑	1.0000	Initial estimates.
32	32.	
f SOLVE A	7.5137	Root found.
g R↓	7.5137	Same previous estimate.
g R↓	0.0000	Function value.
f R↑ f R↑	7.5137	Restore stack.

The day was March 7th. Convert the fractional portion of the number to decimal hours and then to hours, minutes, and seconds.

Keystrokes	Display	
h FRAC	0.5137	Fractional portion of day.
24 ×	12.3293	Decimal hours.
f →H.MS	12.1945	Hours, minutes, seconds.

* Press f FIX 4 to obtain the display settings in this appendix.

Explorer Coulerre saw the sun on March 7th at 12^h 19^m 45^s (Coordinated Universal Time).

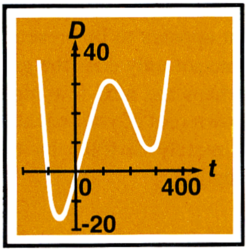
By examining Coulerre's function $f(t)$, you realize that it can have as many as four real roots—three positive and one negative. Try to find additional positive roots by using SOLVE with larger positive estimates.

Keystrokes	Display	
1000 ENTER↑ 1100	**1,100.**	Two larger, positive estimates.
f SOLVE A	**Error 6**	No root found.
CLx	**278.4497**	Last estimate tried.
g R↓	**276.7942**	A previous estimate.
g R↓	**7.8948**	Non-zero value of function.
f R↑ f R↑	**278.4497**	Restore stack to original state.
f SOLVE A	**Error 6**	Again, no root found.
CLx	**278.4398**	Approximately same estimate.
g R↓	**278.4497**	A previous estimate.
g R↓	**7.8948**	Same function value.

You have found a positive local minimum rather than a root. Now try to find the negative root.

Keystrokes	Display	
1000 CHS ENTER↑	**-1,000.0000**	Two larger, negative estimates.
1100 CHS	**-1,100.**	
f SOLVE A	**-108.9441**	Negative root.
g R↓	**-108.9441**	Same previous estimate.
g R↓	**1.6000 -08**	Function value.

There is no need to search further—you have found all possible roots. The negative root has no meaning since it is outside of the range for which the declination approximation is valid. The graph of the function confirms the results you have found.



Graph of *f*(*t*)

Finding Several Roots

Many equations that you encounter have more than one root. For this reason, you will find it helpful to understand some techniques for finding several roots of an equation.

The simplest method for finding several roots is to direct the root search in different ranges of x where roots may exist. Your initial estimates specify the range that is initially searched. This method is used throughout section 8, Finding the Roots of an Equation. You can often find the roots of an equation in this manner.

A more advanced method is know as *deflation*. This technique is useful when the function in an equation has characteristics that make it difficult for SOLVE to find all of the roots. Deflation is a method by which roots are "eliminated" from an equation. This involves modifying the equation so that the first roots found are no longer roots, but the rest of the roots remain roots.

If a function $f(x)$ has a value of zero at $x = a$, then the new function $\frac{f(x)}{(x-a)}$ will not approach zero in this region (if a is a simple root of $f(x) = 0$). You can use this information to eliminate a known root. Simply add a few program lines at the end of your function subroutine. These lines should subtract the known root (to 10 significant digits) from the x value and divide this difference into the function value. In many cases the root will be a simple one, and the new function will direct SOLVE away from the known root.

On the other hand, the root may be a *multiple root*. A multiple root is one that appears to be present repeatedly, in the following sense: at such a root, not only does the graph of $f(x)$ cross the x-axis, but its slope (and perhaps the next few higher-order derivatives) is equal to zero. If the known root of your equation is a multiple root, the root is not eliminated by merely dividing by the factor described above. For example, the equation

$$f(x) = x(x - a)^3 = 0$$

has a multiple root at $x = a$ (with a multiplicity of 3). This root is not eliminated by dividing $f(x)$ by $(x - a)$. But it can be eliminated by dividing by $(x - a)^3$.

Example: Use deflation to help find the roots of

$$60x^4 - 944x^3 + 3003x^2 + 6171x - 2890 = 0.$$

Using Horner's method, this equation can be rewritten in the form

$$(((60x - 944)x + 3003)x + 6171)x - 2890 = 0.$$

Slide the PRGM-RUN switch to PRGM ▮. Key in a subroutine to evaluate the polynomial.

Keystrokes	**Display**
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 2	*001- 25, 13, 2*
6	*002- 6*
0	*003- 0*
$\boxed{\times}$	*004- 61*
9	*005- 9*
4	*006- 4*
4	*007- 4*
$\boxed{-}$	*008- 41*
$\boxed{\times}$	*009- 61*
3	*010- 3*
0	*011- 0*
0	*012- 0*
3	*013- 3*

Keystrokes	Display
+	014- 51
×	015- 61
6	016- 6
1	017- 1
7	018- 7
1	019- 1
+	020- 51
×	021- 61
2	022- 2
8	023- 8
9	024- 9
0	025- 0
−	026- 41
h RTN	027- 25 12

Slide the PRGM-RUN switch to RUN. Key in two large, negative initial estimates (such as −10 and −20) and use SOLVE to find the most negative root.

Keystrokes	Display	
10 CHS ENTER↑	-10.0000	Initial estimates.
20 CHS	-20	
f SOLVE 2	-1.6667	First root.
STO 0	-1.6667	Store root for deflation.
g R↓ g R↓	4.0000 -06	Function value near zero.

Slide the PRGM-RUN switch to PRGM. Add instructions to your subroutine to eliminate the root just found.

Keystrokes	Display	
GTO . 026	026- 41	Line before RTN instruction.
x⪤y	027- 21	Bring x into X-register.
RCL 0	028- 24 0	Divide by $(x - a)$, where a is known root.
−	029- 41	
÷	030- 71	

Now slide the PRGM-RUN switch to RUN. Use the same initial estimates to find the next root.

Keystrokes	Display	
10 $\boxed{\text{CHS}}$ $\boxed{\text{ENTER}\uparrow}$	*-10.0000*	Same initial estimates.
20 $\boxed{\text{CHS}}$	*-20.*	
$\boxed{\text{f}}$ $\boxed{\text{SOLVE}}$ 2	*0.4000*	Second root.
$\boxed{\text{STO}}$ 1	*0.4000*	Store root for deflation.
$\boxed{\text{g}}$ $\boxed{\text{R}\downarrow}$ $\boxed{\text{g}}$ $\boxed{\text{R}\downarrow}$	*0.0000*	Deflated function value.

With the PRGM-RUN switch set to PRGM, modify your subroutine to eliminate the second root.

Keystrokes	Display	
$\boxed{\text{GTO}}$ $\boxed{\cdot}$ 030	*030- 71*	Line before $\boxed{\text{RTN}}$ instruction.
$\boxed{x\gtrless y}$	*031- 21*	Bring x into X-register.
$\boxed{\text{RCL}}$ 1	*032- 24 1*	Deflation for second root.
$\boxed{-}$	*033- 41*	
$\boxed{\div}$	*034- 71*	

Slide the PRGM-RUN switch to RUN. Again, use the same initial estimates to find the next root.

Keystrokes	Display	
10 $\boxed{\text{CHS}}$ $\boxed{\text{ENTER}\uparrow}$	*-10.0000*	Same initial estimates.
20 $\boxed{\text{CHS}}$	*-20*	
$\boxed{\text{f}}$ $\boxed{\text{SOLVE}}$ 2	*8.4999*	Third root.
$\boxed{\text{STO}}$ 2	*8.4999*	Store root for deflation.
$\boxed{\text{g}}$ $\boxed{\text{R}\downarrow}$ $\boxed{\text{g}}$ $\boxed{\text{R}\downarrow}$	*-1.0929 -07*	Deflated function value near zero.

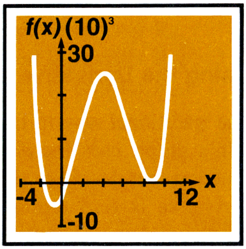
With the PRGM-RUN switch set to PRGM, change your subroutine to eliminate the third root.

Keystrokes	Display		
GTO · 034	034-	71	Line before RTN instruction.
x⇄y	035-	21	Bring x into X-register.
RCL 2	036-	24 2	Deflation for third root.
−	037-	41	
÷	038-	71	

Slide the PRGM-RUN switch to RUN and find the fourth root.

Keystrokes	Display	
10 CHS ENTER↑	-10.0000	Same initial estimates.
20 CHS	-20.	
f SOLVE 2	8.5001	Fourth root.
STO 3	8.5001	Store root for reference.
g R↓ g R↓	-0.0009	Deflated function value near zero.

Using the same initial estimates each time, you have found four roots for this equation involving a fourth-degree polynomial. However, the last two roots are quite close to each other, and are actually one root (with a multiplicity of 2). That is why the root was not eliminated when you tried deflation once at this root. (Round-off error causes the original function to have small positive and negative values for values of x between 8.4999 and 8.5001; for $x = 8.5$ the function is exactly zero.)



Graph of $f(x)$

In general, you will not know in advance the multiplicity of the root you are trying to eliminate. If, after you have attempted to eliminate a root, SOLVE finds that same root again, you can proceed in a number of ways:

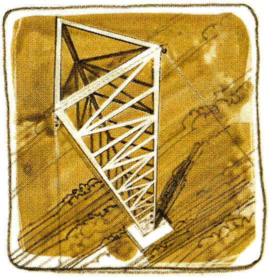
- Use different initial estimates with the *deflated* function in an attempt to search for a different root.
- Use deflation again in an attempt to eliminate a multiple root. If you do not know the multiplicty of the root, you may need to repeat this a number of times.
- Examine the behavior of the *deflated* function at x values near the known root. If the function's calculated values cross the x-axis smoothly, either another root or a greater multiplicity is indicated.
- Analyze the original function algebraically. It may be possible to determine its behavior for x values near the known root. (A Taylor series representation, for example, may indicate the multiplicity of a root.)

Finding Local Extremes of a Function

Using the Derivative

The traditional way to find local maximums and minimums of a function's graph uses the *derivative* of the function. The derivative is a function that describes the slope of the graph. Values of x at which the derivative is zero represent potential local extremes of the function. (Although less common for well-behaved functions, values of x where the derivative is infinite or undefined are also possible extremes.) If you can express the derivative of a function in closed form, you can use SOLVE to find where the derivative is zero—showing where the function may be maximum or minimum.

Example: For the design of a vertical broadcasting tower, radio engineer Ann Tenor wants to find the angle from the tower at which the relative field intensity is most negative. The relative intensity created by the tower is given by



$$E = \frac{\cos(2\pi h \cos\theta) - \cos(2\pi h)}{[1 - \cos(2\pi h)] \sin\theta}$$

where E is the relative field intensity, h is the antenna height in wavelengths, and θ is the angle from vertical in radians. The height is 0.6 wavelengths for her design.

Solution: The desired angle is one at which the derivative of the intensity with respect to θ is zero.

To save program memory space and execution time, store the following constants in registers and recall them as needed:

$R0 = 2\pi h$	and is stored in register R_0,
$R1 = \cos(2\pi h)$	and is stored in register R_1,
$R2 = 1/[1-\cos(2\pi h)]$	and is stored in register R_2.

The derivative of the intensity E with respect to the angle θ is given by

$$\frac{dE}{d\theta} = R2\left[R0 \sin(R0\cos\theta) - \frac{\cos(R0\cos\theta) - R1}{\sin\theta\tan\theta}\right].$$

Slide the PRGM-RUN switch to PRGM and key in a subroutine to calculate the derivative.

Keystrokes	Display
f CLEAR PRGM	000-
h LBL 0	001- 25, 13, 0
f COS	002- 14 8
RCL 0	003- 24 0
×	004- 61
f COS	005- 14 8
RCL 1	006- 24 1
−	007- 41
x⪤y	008- 21
f SIN	009- 14 7
÷	010- 71
x⪤y	011- 21
f TAN	012- 14 9
÷	013- 71
CHS	014- 32
x⪤y	015- 21
f COS	016- 14 8
RCL 0	017- 24 0
×	018- 61
f SIN	019- 14 7
RCL 0	020- 24 0
×	021- 61
+	022- 51
RCL 2	023- 24 2
×	024- 61
h RTN	025- 25 12

Now slide the PRGM-RUN switch to RUN. In radian mode, calculate and store the three constants.

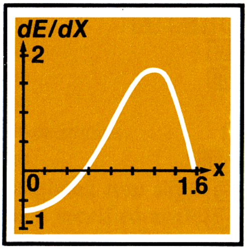
Keystrokes	Display	
g RAD	0.0000	Specify radian mode. (Assumes display has been cleared.)

Keystrokes	Display	
2 h π ×	6.2832	
.6 × STO 0	3.7699	Constant $R0$.
f COS STO 1	-0.8090	Constant $R1$.
CHS 1 +	1.8090	
h 1/x STO 2	0.5528	Constant $R2$.

The relative field intensity is maximum at an angle of 90° (perpendicular to the tower). To find the minimum, use angles closer to zero as initial estimates, such as the radian equivalents of 10° and 60°.

Keystrokes	Display		
10 g →R	0.1745		Initial estimates.
60 g →R	1.0472		
f SOLVE 0	0.4899		Angle giving zero slope.
g R↓ g R↓	-5.5279	-10	Slope at specified angle.
f R↑ f R↑	0.4899		Restore the stack.
f →D	28.0680		Angle in degrees.

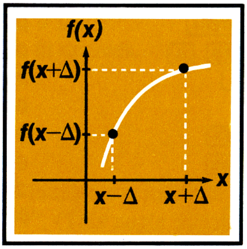
The relative field intensity is most negative at an angle of 28.0680° from vertical.



Graph of $dE/d\theta$ Versus θ

Using an Approximate Slope

The derivative of a function can also be approximated numerically. If you sample a function at two points relatively close to x (namely $x+\Delta$ and $x-\Delta$), you can calculate an average slope of the function's graph



$$s = \frac{f(x+\Delta) - f(x-\Delta)}{2\Delta}.$$

The accuracy of this approximation depends upon the increment Δ and the nature of the function. Smaller values of Δ give better approximations to the derivative, but excessively small values can cause round-off inaccuracy. A value of x at which the slope is zero is potentially a local extreme of the function.

Example: Solve the previous example without using the equation for the derivative $dE/d\theta$.

Solution: Find the angle at which the derivative (determined numerically) of the intensity E is zero.

Slide the PRGM-RUN switch to PRGM and key in two subroutines: one to estimate the derivative of the intensity and one to evaluate the intensity function E. In the following subroutine, the slope is calculated between $\theta + 0.001$ and $\theta - 0.001$ radians (a range equivalent to approximately 0.1°).

Keystrokes	Display	
h LBL A	001– 25, 13, 11	
EEX	002– 33	Evaluate E at $\theta + 0.001$.
CHS	003– 32	
3	004– 3	
+	005– 51	
ENTER↑	006– 31	
GSB B	007– 13 12	
x⇄y	008– 21	
EEX	009– 33	Evaluate E at $\theta - 0.001$.
CHS	010– 32	
3	011– 3	
−	012– 41	
ENTER↑	013– 31	
GSB B	014– 13 12	
−	015– 41	
2	016– 2	
EEX	017– 33	
CHS	018– 32	
3	019– 3	
÷	020– 71	
h RTN	021– 25 12	

Keystrokes	Display
h LBL B	*022– 25, 13, 12*
f COS	*023– 14 8*
RCL 0	*024– 24 0*
×	*025– 61*
f COS	*026– 14 8*
RCL 1	*027– 24 1*
−	*028– 41*
x⇄y	*029– 21*
f SIN	*030– 14 7*
÷	*031– 71*
RCL 2	*032– 24 2*
×	*033– 61*
h RTN	*034– 25 12*

Slide the PRGM-RUN switch to ▮▥RUN. In the previous example, the calculator was set to radian mode and the three constants were stored in registers 0, 1, and 2. Key in the same initial estimates as before and execute SOLVE.

Keystrokes	Display	
10 g →R	*0.1745*	Initial estimates.
60 g →R	*1.0472*	
f SOLVE A	*0.4899*	Angle giving zero slope.
g R↓ g R↓	*0.0000*	Slope at specified angle.
f R↑ f R↑	*0.4899*	Restore the stack.
ENTER↑ ENTER↑ B	*−0.2043*	Use function subroutine to calculate minimum intensity.
x⇄y	*0.4899*	Recall θ value.
f →D	*28.0679*	Angle in degrees.

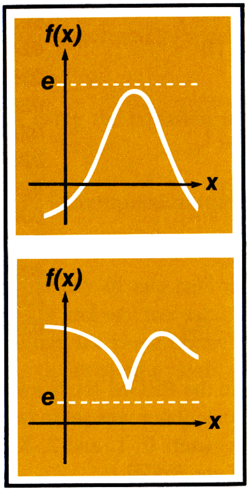
This numerical approximation of the derivative indicates a minimum field intensity of −0.2043 at an angle of 28.0679°. (This angle differs from the previous solution by 0.0001°.)

Using Repeated Estimation

A third technique is useful when it is not practical to calculate the derivative. It is a slower method because it requires the repeated use of the SOLVE key. On the other hand, you do not have to find a good value for Δ of the previous method. To find a local extreme of the function $f(x)$, define a new function

$$g(x) = f(x) - e$$

where e is a number slightly beyond the estimated extreme value of $f(x)$. If e is properly chosen, $g(x)$ will *approach* zero near the extreme of $f(x)$ but will not *equal* zero. Use SOLVE to analyze $g(x)$ near the extreme. The desired result is ***Error 6.***



- If ***Error 6*** is displayed the number in the X-register is an x value near the extreme. The number in the Z-register tells roughly how far e is from the extreme value of $f(x)$. Revise e to bring it closer (but not equal) to the extreme value. Then use SOLVE to examine the revised $g(x)$ near the x value previously found. Repeat this procedure until successive x values do not differ significantly.

- If a root of $g(x)$ is found, either the number e is *not* beyond the extreme value of $f(x)$ or else SOLVE has found a different region where $f(x)$ equals e. Revise e so that it is close to—but beyond—the extreme value of $f(x)$ and try SOLVE again. It may also be possible to modify $g(x)$ in order to eliminate the distant root.

Example: Solve the previous example without calculating the derivative of the relative field intensity *E*.

Solution: The subroutine to calculate *E* and the required constants have been entered in the previous examples.

Slide the PRGM-RUN switch to PRGM ▥█ . Key in a subroutine which subtracts an estimated extreme number from the field intensity *E*. The extreme number should be stored in a register so that it can be manually changed as needed.

Keystrokes	**Display**	
$\boxed{\text{h}}$ $\boxed{\text{LBL}}$ 1	*001– 25, 13, 1*	Begin with $\boxed{\text{LBL}}$ instruction.
$\boxed{\text{GSB}}$ $\boxed{\text{B}}$	*002– 13 12*	Calculate *E*.
$\boxed{\text{RCL}}$ 9	*003– 24 9*	Subtract extreme estimate.
$\boxed{-}$	*004– 41*	
$\boxed{\text{h}}$ $\boxed{\text{RTN}}$	*005– 25 12*	

Slide the PRGM-RUN switch to █▥ RUN . Estimate the minimum intensity value by manually sampling the function.

Keystrokes	**Display**	
10 $\boxed{\text{g}}$ $\boxed{\to\text{R}}$	*0.1745*	Sample the function at 10°, 30°, 50°, …
$\boxed{\text{ENTER}\uparrow}$ $\boxed{\text{B}}$	*-0.1029*	
30 $\boxed{\text{g}}$ $\boxed{\to\text{R}}$	*0.5236*	
$\boxed{\text{ENTER}\uparrow}$ $\boxed{\text{B}}$	*-0.2028*	
50 $\boxed{\text{g}}$ $\boxed{\to\text{R}}$	*0.8727*	
$\boxed{\text{ENTER}\uparrow}$ $\boxed{\text{B}}$	*0.0405*	

Based on these samples, try using an extreme estimate of −0.25 and initial SOLVE estimates (in radians) near 10° and 30°.

Keystrokes	Display	
.25 CHS STO 9	−0.2500	Store extreme estimate.
.2 ENTER↑	0.2000	Initial estimates.
.6	0.6	
f SOLVE 1	Error 6	No root found.
CLx STO 4	0.4849	Store θ estimate.
g R↓ STO 5	0.4698	Store previous θ estimate.
g R↓	0.0457	Distance from extreme.
.9 ×	0.0411	Revise extreme estimate by 90 percent of the distance.
STO + 9	0.0411	
RCL 4	0.4849	Recall θ estimate.
ENTER↑ ENTER↑ B	−0.2043	Calculate intensity E.
CLx	0.0000	Recall other θ estimate, keeping first estimate in Y-register.
RCL 5	0.4698	
f SOLVE 1	Error 6	No root found.
CLx	0.4898	θ estimate.
x≷y	0.4893	Previous θ estimate.
x≷y	0.4898	Recall θ estimate.
ENTER↑ ENTER↑ B	−0.2043	Calculate intensity E.
x≷y	0.4898	Recall θ value.
f →D	28.0660	Angle in degrees.

The second iteration produces two θ estimates that differ in the fourth decimal place. The field intensities E for the two iterations are equal to four decimal places. Stopping at this point, a minimum field intensity of −0.2043 is indicated at an angle of 28.0660°. (This angle differs from the previous solutions by about 0.002°.)

Limiting the Estimation Time

Occasionally, you may desire to limit the time used by SOLVE to find a root. You can use two possible techniques to do this—counting iterations and specifying a tolerance.

Counting Iterations

While searching for a root, SOLVE typically samples your function at least a dozen times. Occasionally, SOLVE may need to sample it one hundred times or more. (However, SOLVE will always stop by itself.) Because your function subroutine is executed once for each estimate that is tried, it can count and limit the number of iterations. An easy way to do this is with an ISG instruction to accumulate the number of iterations in the I-register. If you store an appropriate number in the I-register before using SOLVE, your subroutine can interrupt the SOLVE algorithm when the limit is exceeded. The ISG instruction is discussed on page 141.)

Specifying a Tolerance

You can shorten the time required to find a root by specifying a tolerable inaccuracy for your function. Your subroutine should return a function value of zero if the calculated function value is less than the specified tolerance. This tolerance that you specify should correspond to a value that is negligible for practical purposes or should correspond to the accuracy of the computation. This technique eliminates the time required to define the estimate more accurately than is justified by the problem. (Example of this method are given on page 190 and below.)

Using SOLVE With $\int_x^y$

Example: For a phase-modulated radio signal, the amplitude of the carrier signal is proportional to $J_0(x)$, the zero-order Bessel function of the first kind, where x is the modulation index. What is the smallest modulation index at which the carrier signal is suppressed (that is, its amplitude is zero)?

Solution: The desired index is the smallest value of x for which

$$J_0(x) = \int_0^\pi \frac{\cos(x \sin \theta)}{\pi} \, d\theta = 0.$$

You can use SOLVE to determine this value. The function $J_0(x)$ must be calculated by using $\int_x^y$.

The approximation of $J_0(x)$ calculated by $\int_y^x$ has an uncertainty that is returned in the Y-register. Whenever the magnitude of $J_0(x)$ is less than this uncertainty, $J_0(x)$ can be considered to be zero. By using this technique, you can prevent SOLVE from seeking unreasonable accuracy.

Slide the PRGM-RUN switch to PRGM. Key in a subroutine that calculates $J_0(x)$ and a subroutine that calculates the function to be integrated.

Keystrokes	**Display**	
f CLEAR PRGM	000-	Clear program memory.
h LBL A	001- 25, 13, 11	Begin with LBL instruction.
STO 0	002- 23 0	Store argument x.
0	003- 0	Limits of integration.
h π	004- 25 73	
f $\int_y^x$ 3	005- 14, 72, 3	Calculate $J_0(x)$.
h ABS	006- 25 34	Magnitude of $J_0(x)$.
f $x \leq y$	007- 14 41	Return zero if $J_0(x) \leq$ uncertainty.
CLx	008- 34	
g $x \neq 0$	009- 15 61	Restore $J_0(x)$ if value is nonzero.
h LST x	010- 25 0	
h RTN	011- 25 12	
h LBL 3	012- 25, 13, 3	Calculate function to be integrated.
f SIN	013- 14 7	
RCL 0	014- 24 0	
×	015- 61	
f COS	016- 14 8	
h π	017- 25 73	
÷	018- 71	
h RTN	019- 25 12	

In order to shorten the time to find the desired root, initially specify SCI 0 display mode for the integration. After an approximate solution has been found, specify a greater integration accuracy (by using SCI 3). Then let SOLVE home in on the root using the more accurate function. This procedure eliminates the need to integrate with great accuracy for values of x not near the root, saving considerable time.

Slide the PRGM-RUN switch to RUN and perform the following steps. Keep in mind that $\boxed{\text{SOLVE}}$ samples your function many times and that $\boxed{\int_y^x}$ often requires up to a minute or more to evaluate an integral. For these reasons the $\boxed{\text{SOLVE}}$ executions that follow take about 3 and 6 minutes to be completed.

Keystrokes	**Display**		
$\boxed{f}$ $\boxed{\text{SCI}}$ 0	***0.***	***00***	Specify the $\boxed{\int_y^x}$ accuracy. (Assumes that the display has been cleared.)
$\boxed{g}$ $\boxed{\text{RAD}}$	***0.***	***00***	
0 $\boxed{\text{ENTER}\uparrow}$	***0.***	***00***	Initial estimates to
1	***1.***		search near 0.
$\boxed{f}$ $\boxed{\text{SOLVE}}$ $\boxed{A}$	***2.***	***00***	Desired root.
$\boxed{f}$ $\boxed{\text{SCI}}$ 3	***2.480***	***00***	Specify greater $\boxed{\int_y^x}$ accuracy.
2.4 $\boxed{\text{ENTER}\uparrow}$	***2.400***	***00***	Initial estimates near
2.5	***2.5***		first approximation.
$\boxed{f}$ $\boxed{\text{SOLVE}}$ $\boxed{A}$	***2.405***	***00***	Desired root.
$\boxed{f}$ $\boxed{\text{FIX}}$ 4	***2.4049***		View in $\boxed{\text{FIX}}$ 4 format.
$\boxed{g}$ $\boxed{R\downarrow}$ $\boxed{g}$ $\boxed{R\downarrow}$	***0.0000***		$J_0(x)$ is less than uncertainty.
$\boxed{g}$ $\boxed{R\downarrow}$	***0.0001***		Uncertainty from $\boxed{\int_y^x}$.

A modulation index of 2.4049 causes the carrier signal amplitude to be suppressed by at least 99.99%. (That is, its amplitude is less than 0.0001 of maximum.)

Appendix B

A More Detailed Look at $\boxed{\int_x^y}$

Section 9 presented the basic information you need to use $\boxed{\int_x^y}$ knowledgeably in most applications. This appendix discusses more esoteric aspects of $\boxed{\int_x^y}$ that may be of interest to you if you use $\boxed{\int_x^y}$ often.

How $\boxed{\int_x^y}$ Works

The $\boxed{\int_x^y}$ algorithm 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: 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 $f(x)$—provide an exact answer.

Evaluating the function at an infinite number of sample points would take a very long time (namely, forever). Fortunately, 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 $\boxed{\int_x^y}$ 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 (that is, 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 $f(x)$.

The uncertainty of the final approximation is a number derived from the display format, which indicates the uncertainty in the function.* At the end of each iteration, the algorithm compares the approximation calculated during that iteration with the approximations calculated during two

* The relationship between the display format, the uncertainty in the function, and the uncertainty in the approximation to its integral are discussed later in this appendix.

previous iterations. If the difference between any of these three approximations and the other two is less than the uncertainty of the final approximation, the algorithm terminates, placing the current approximation in the X-register and its uncertainty in the Y-register.

The $\boxed{\int_y^x}$ algorithm is designed so that it is extremely unlikely that the error in each of three successive approximations—that is, the differences between the actual integral and the approximations—would all be less than the disparity among the approximations themselves. Consequently, the error in the final approximation will be less than its uncertainty.* Although we can't know the error in the final approximation, we *can* be very confident that the error is less than the displayed uncertainty of the approximation. Thus, the uncertainty of the approximation is an ''upper bound'' on the difference between the approximation and the actual integral.

Accuracy, Uncertainty, and Calculation Time

The accuracy of an $\boxed{\int_y^x}$ approximation does not always change when you increase *by just one* the number of digits specified in the display format. Similarly, the time required to calculate an integral sometimes changes when you change the display format, but sometimes does not.

Example: The Bessel function of the first kind of order four can be expressed as

$$J_4(x) = \frac{1}{\pi}\int_0^\pi \cos(4\theta - x\sin\theta)\,d\theta.$$

Calculate the integral in the expression for $J_4(1)$,

$$\frac{1}{\pi}\int_0^\pi \cos(4\theta - \sin\theta)\,d\theta.$$

* Provided that $f(x)$ is sufficiently smooth, a consideration we will discuss in more detail later in this appendix.

First, slide the PRGM-RUN switch to PRGM ▮▮, and key in a subroutine that evaluates the function $f(\theta) = \cos(4\theta - \sin\theta)$.

Keystrokes	Display
f CLEAR PRGM	000–
h LBL 0	001– 25, 13, 0
4	002– 4
×	003– 61
x⇄y	004– 21
f SIN	005– 14 7
−	006– 41
f COS	007– 14 8
h RTN	008– 25 12

Now, slide the PRGM-RUN switch back to ▮▮ RUN, and key the limits of integration into the X- and Y-registers. Ensure that the trigonometric mode is set to radians, and set the display format to SCI 2. Finally, press f ∫ₓʸ 0 to calculate the integral.

Keystrokes	Display	
0 ENTER↑	0.0000	Key lower limit into Y-register.
h π	3.1416	Key upper limit into X-register.
g RAD	3.1416	Ensure that trigonometric mode is set to radians. (This step is not necessary if you have not switched your calculator off nor reset the trigonometric mode since you last set it to radians.)
f SCI 2	3.14 00	Set display format to SCI 2.
f ∫ₓʸ 0	7.79 -03	Integral approximated in SCI 2.
x⇄y	1.45 -03	Uncertainty of SCI 2 approximation.

The uncertainty indicates that the displayed digits of the approximation might not include any digits that could be considered accurate. Actually, just like the last approximation in section 9, this approximation is more accurate than its uncertainty indicates.

Keystrokes	**Display**	
x≷y	*7.79 -03*	Return approximation to display.
h MANT	*7785820888*	All 10 digits of SCI 2 approximation.

The actual value of this integral, correct to five significant digits, is 7.7805×10^{-3}. Therefore, the error in this approximation is about $(7.7858 - 7.7805) \times 10^{-3} = 5.3 \times 10^{-6}$. This error is considerably less than the uncertainty, 1.45×10^{-3}. The uncertainty is only an *upper bound* on the error in the approximation; the actual error will generally be smaller.

Now let's calculate the integral in SCI 3 and compare the accuracy of the resulting approximation to that of the SCI 2 approximation.

Keystrokes	**Display**	
f SCI 3	*7.786 -03*	Change display format to SCI 3.
g R↓ g R↓	*3.142 00*	Roll down stack until upper limit appears in X-register.
f ∫ʸₓ 0	*7.786 -03*	Integral approximated in SCI 3.
x≷y	*1.448 -04*	Uncertainty of SCI 3 approximation.
x≷y	*7.786 -03*	Return approximation to display.
h MANT	*7785820888*	All 10 digits of SCI 3 approximation.

All 10 digits of the approximations in SCI 2 and in SCI 3 are identical: the accuracy of the approximation in SCI 3 is no better than the accuracy in SCI 2, despite the fact that the uncertainty in SCI 3 is less than the uncertainty in SCI 2. Why is this? Remember that the accuracy of any approximation depends primarily on the number of sample points at which the function $f(x)$ has been evaluated. The ∫ₓʸ algorithm is iterated with increasing numbers of sample points until the disparity among three successive approximations is less than the uncertainty, which is a number derived from the display format. After a particular iteration, the disparity among the approximations may already be so much less than the uncertainty that it would still be less if the uncertainty were decreased by a factor of 10. In such cases, if you decreased the uncertainty by specifying one more digit in the display format, the algorithm would not have to consider additional sample points, and the resulting approximation would be identical to the approximation calculated with the larger uncertainty.

If you calculated the two preceding approximations on your calculator, you may have noticed that it took just as long to calculate the integral in SCI 3 as in SCI 2. This is because the time to calculate the integral of a given function depends on the number of sample points at which the function must be evaluated to achieve an approximation of acceptable accuracy. For the SCI 3 approximation, the algorithm did not have to consider more sample points that it did in SCI 2, so it did not take any longer to calculate the integral.

Often, however, increasing the number of digits in the display format will require evaluating the function at additional sample points, so that calculating the integral will take more time. Let's now calculate the same integral in SCI 4:

Keystrokes	Display	
f SCI 4	7.7858 -03	Change display format to SCI 4.
g R↓ g R↓	3.1416 00	Roll down stack until upper limit appears in X-register.
f ∫ₓʸ 0	7.7807 -03	Integral approximated in SCI 4.

This approximation took about twice as long as the approximation in SCI 3 or SCI 2. In this case, the algorithm had to evaluate the function at about twice as many sample points as before in order to achieve an approximation of acceptable accuracy. Note, however, that we received a reward for our patience: the accuracy of this approximation is better, by almost two digits, than the accuracy of the approximation calculated using half the number of sample points.

The preceding examples show that repeating the approximation of an integral in a different display format sometimes will give you a more accurate answer, but sometimes it will not. Whether or not the accuracy is changed depends on the particular function, and generally can be determined only by trying it.

Furthermore, if you do get a more accurate answer, it will come at the cost of about double the calculation time. This unavoidable trade-off between accuracy and time is important to keep in mind if you are considering decreasing the uncertainty in hopes of obtaining a more accurate answer.

Note: The time required to calculate the integral of a given function depends not only on the number of digits specified in the display format, but also, to a certain extent, on the limits of integration. When the calculation of an integral requires an excessive amount of time, the width of the interval of integration (that is, the difference of the limits) may be too large compared with certain features of the function being integrated. For most problems, however, you need not be concerned about the effects of the limits of integration on the calculation time. These considerations, together with examples where the limits may be unduly prolonging the calculation time as well as techniques for dealing with such situations, will be discussed later in this appendix.

Accuracy of the Function to be Integrated

The accuracy of an integral calculated using ∫ⁿ depends on the accuracy of the function calculated by your subroutine. This accuracy, which you specify using the display format, depends primarily on three considerations:

1. The accuracy of empirical constants in the function.

2. The degree to which the function may accurately describe a physical situation.
3. The extent of round-off error in the internal calculations of the calculator.

Functions Related to Physical Situations

The functions we've integrated so far in section 9 and this appendix—$\cos(\sin\theta)$, $\cos(\theta - \sin\theta)$, $\cos(4\theta - \sin\theta)$, and $(\sin x)/x$—are examples of *pure mathematical functions*. In this context, this means that the functions do not contain any empirical constants, and neither the variables nor the limits of integration represent actual physical quantities. For such functions, you can specify as many digits as you want in the display format (up to nine) to achieve the desired degree of accuracy in the integral.* All you need to consider is the trade-off between the accuracy and calculation time.

There are additional considerations, however, when you're integrating functions relating to an actual physical situation. Basically, with such functions you should ask yourself *whether the accuracy you would like in the integral is justified by the accuracy in the function*. For example, if the function contains empirical constants that are specified to only, say, three significant digits, it might not make sense to specify more than three digits in the display format.

Another important consideration—and one which is more subtle and therefore more easily overlooked—is that nearly every function relating to a physical situation *is inherently inaccurate to a certain degree*, because it is only a mathematical *model* of an actual process or event. A mathematical model is itself an *approximation* that ignores the effects of known or unknown factors which are insignificant to the degree that the results are still useful.

An example of a mathematical model is the *normal distribution function*

$$\int_{-\infty}^{t} \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}\,dx,$$

* Provided that $f(x)$ is still calculated accurately, despite round-off error, to the number of digits shown in the display.

which has been found to be useful in deriving information concerning physical measurements on living organisms, product dimensions, average temperatures, etc. A similar mathematical model is

$$C = \frac{C_0}{\sqrt{\pi D t}} \int_{x/2\sqrt{Dt}}^{\infty} e^{-y^2/4Dt}\, dy,$$

which is a particular solution of the diffusion equation for semiconductors. Such mathematical descriptions typically are either derived from theoretical considerations or inferred from experimental data. To be practically useful, they are constructed with certain assumptions, such as ignoring the effects of relatively insignificant factors. For example, the accuracy of results obtained using the normal distribution function as a model of the distribution of certain quantities depends on the size of the population being studied. The accuracy of results obtained using the solution to the diffusion equation ignores quantum effects. And the accuracy of results obtained from the equation $s = s_0 - \frac{1}{2}gt^2$, which gives the height of a falling body, ignores the variation with altitude of g, the acceleration of gravity.

Thus, mathematical descriptions of the physical world can provide results of only limited accuracy. If numerical results of the model are needed to only, say, three significant digits, the effects of many factors and assumptions can be ignored. On the other hand, such factors and assumptions might, if they could be included in a more precise mathematical description—which would still be only a model—affect the digits in the fifth and succeeding decimal places. If you calculated an integral with an apparent accuracy beyond that with which the model describes the actual behavior of the process or event, you would not be justified in using the calculated value to the full apparent accuracy.

Round-Off Error in Internal Calculations

With any computational device—including your HP-34C—calculated results must be "rounded off" to a finite number of digits (10 digits in your HP-34C). Because of this *round-off error,* calculated results—especially results of evaluating a function that contains several mathematical operations—may not be accurate to all 10 digits that can be displayed. Note that round-off error affects the evaluation of *any* mathematical expression, not just the evaluation of a function to be integrated using $\boxed{\int_y^x}$.

If $f(x)$ is a function relating to a physical situation, its inaccuracy due to round-off typically is insignificant compared to the inaccuracy due to empirical constants, etc. If $f(x)$ is what we have called a pure mathematical function, its accuracy is limited only by round-off error. Generally, it would require a complicated analysis to determine precisely how many digits of a calculated function might be affected by round-off. In practice, its effects are typically (and adequately) determined through experience rather than analysis.

In certain situations round-off error can cause peculiar results, particularly if you should compare the results of calculating integrals that are equivalent mathematically but differ by a transformation of variables. Describing such situations—which you are unlikely to encounter in typical applications—is beyond the scope of this handbook.

Uncertainty and the Display Format

Because of round-off error, the subroutine you write for evaluating $f(x)$ cannot calculate $f(x)$ exactly, but rather calculates

$$\hat{f}(x) = f(x) \pm \delta_1(x),$$

where $\delta_1(x)$ is the uncertainty of $f(x)$ caused by round-off error.

If $f(x)$ relates to a physical situation, then the function you would like to integrate is not $f(x)$ but rather

$$F(x) = f(x) \pm \delta_2(x),$$

where $\delta_2(x)$ is the uncertainty associated with $f(x)$ that is caused by the approximation to the actual physical situation.

Since $f(x) = \hat{f}(x) \pm \delta_1(x)$, the function you want to integrate is

$$F(x) = \hat{f}(x) \pm \delta_1(x) \pm \delta_2(x)$$

or

$$F(x) = \hat{f}(x) \pm \delta(x),$$

where $\delta(x)$ is the net uncertainty associated with $\hat{f}(x)$.

Therefore, the integral you want is

$$\begin{aligned}\int_a^b F(x)\,dx &= \int_a^b \left[\hat{f}(x) \pm \delta(x)\right] dx \\ &= \int_a^b \hat{f}(x)\,dx \pm \int_a^b \delta(x)\,dx \\ &= I \pm \Delta\end{aligned}$$

where I is the approximation to $\int_a^b F(x)\,dx$ and Δ is the uncertainty associated with the approximation. The $\boxed{\int_y^x}$ algorithm places the number I in the X-register and the number Δ in the Y-register.

The uncertainty $\delta(x)$ of $\hat{f}(x)$, the function calculated by your subroutine, is determined as follows. Suppose you consider three significant digits of the function's values to be accurate, so you set the display format to $\boxed{\text{SCI}}$ 2. The display would then show only the accurate digits in the mantissa of a function's values: for example, ***1.23 -04***.

Since the display format rounds the number in the X-register to the number displayed, this implies that the uncertainty in the function's values is $\pm 0.005 \times 10^{-4} = \pm 0.5 \times 10^{-2} \times 10^{-4} = \pm 0.5 \times 10^{-6}$. Thus, setting the display format to $\boxed{\text{SCI}}$ n or $\boxed{\text{ENG}}$ n, where n is an integer, implies that the uncertainty in the function's values is

$$\begin{aligned}\delta(x) &= 0.5 \times 10^{-n} \times 10^{m(x)} \\ &= 0.5 \times 10^{-n+m(x)}\end{aligned}$$

In this formula, n is the number of digits specified in the display format and $m(x)$ is the exponent of the function's value at x that would appear if the value were displayed in $\boxed{\text{SCI}}$ display format.

The uncertainty is proportional to the factor $10^{m(x)}$, which represents the magnitude of the function's value at x. Therefore, $\boxed{\text{SCI}}$ and $\boxed{\text{ENG}}$ display formats imply an uncertainty in the function that is *relative* to the function's magnitude.

Similarly, if a function value is displayed in FIX *n*, the rounding of the display implies that the uncertainty in the function's values is

$$\delta(x) = 0.5 \times 10^{-n}.$$

Since this uncertainty is independent of the function's magnitude, FIX display format implies an uncertainty that is *absolute*.

Each time the ∫y x algorithm samples the function at a value of x, it also derives a sample of $\delta(x)$, the uncertainty of the function's value at x. This is calculated using the number of digits n currently specified in the display format and (if the display format is set to SCI or ENG) the magnitude $m(x)$ of the function's value at x. The number Δ, the uncertainty of the approximation to the desired integral, is the integral of $\delta(x)$:

$$\Delta = \int_a^b \delta(x)\, dx$$

$$= \int_a^b \left[0.5 \times 10^{-n+m(x)}\right] dx.$$

This integral is calculated using the samples of $\delta(x)$ in roughly the same way that the approximation to the integral of the function is calculated using the samples of $\hat{f}(x)$.

Because Δ is proportional to the factor 10^{-n}, the uncertainty of an approximation changes by about a factor of 10 for each digit specified in the display format. This will generally not be exact in SCI or ENG display format, however, because changing the number of digits specified may require that the function be evaluated at different sample points, so that $\delta(x) \sim 10^{m(x)}$ would have different values.

Note that when an integral is approximated in FIX display format, $m(x) = 0$ and so the calculated uncertainty in the approximation turns out to be

$$\Delta = 0.5 \times 10^{-n}(b - a).$$

Normally you do not have to determine precisely the uncertainty in the function. (To do so would frequently require a very complicated analysis.) Generally, it's more convenient to use SCI or ENG display format if the uncertainty in the function's values can be more easily estimated as a *relative* uncertainty. On the other hand, it's more convenient to use FIX display format if the uncertainty in the function's values can be more easily estimated as an *absolute* uncertainty. FIX display format may be inappropriate to use (leading to peculiar results) when you are integrating a function whose magnitude *and* uncertainty have extremely small values throughout the interval of integration, or a function whose magnitude and uncertainty vary through extremely large *and* small values within the interval of integration. Likewise, SCI display format may be inappropriate to use (also leading to peculiar results) if the magnitude of the function becomes much smaller than its uncertainty. If the results of calculating an integral seem strange, it may be more appropriate to calculate the integral in the alternate display format.

Calculating Integrals of Maximum Accuracy

In SCI or ENG display format, numbers can be displayed with a mantissa containing up to seven digits. Specifying SCI 8 or SCI 9 generally results in the same *display* as SCI 7. However, the uncertainty of integrals calculated in SCI 8 or SCI 9 is smaller than the uncertainty of integrals calculated in SCI 7. The same is true, of course, for integrals calculated in ENG display format.

You can calculate an integral of greatest possible accuracy with the display mode set to SCI (or ENG) 9.* If the calculator is in RUN mode, you can do so either *directly* by pressing f SCI 9, or *indirectly* by pressing 9 STO f I h DSP I (when the display format is already set to SCI or ENG). If the calculator is in PRGM mode, however, you cannot set the display mode *directly* to SCI 8, SCI 9, ENG 8, or ENG 9. If you attempt to do so, the resulting keycode will indicate SCI 7 or ENG 7, and integrals will be calculated with an uncertainty derived from a

* Provided, of course, that $f(x)$ is calculated accurately to 10 significant digits.

display format specifying seven digits. To calculate integrals of maximum accuracy in PRGM mode, therefore, you must set the display format *indirectly* using DSP I.*

To see how this is done, slide the PRGM-RUN switch to PRGM and key in the following (trivial) program, which calculates the integral of (sin x)/x with maximum accuracy. Afterwards, we'll execute the program to calculate $Si\,(2)$.

Keystrokes	**Display**	
h LBL A	001– 25, 13, 11	Label of program containing $\int_x^y$ in program line.
9	002– 9	Key 9 into X-register.
STO f I	003– 23, 14, 23	Store 9 in I-register.
h DSP I	004– 25 11	Sets display format to nine digits. (This program assumes that the display format will have been manually set to SCI before the program is executed.)
g R↓	005– 15 22	Roll down the stack so that the 9 entered into the X-register in program line 002 does not become the upper limit of integration.
f $\int_x^y$ 2	006– 14, 72, 2	Calculate the integral $\int_a^b (\sin x)/x\, dx$.
h RTN	007– 25 12	
h LBL 2	008– 25, 13, 2	Label of subroutine that evaluates $f(x) = (\sin x)/x$.
f SIN	009– 14 7	
x⪤y	010– 21	
÷	011– 71	
h RTN	012– 25 12	

* If there is a negative number in the I-register when you press h DSP I, numbers will be *displayed* as they would appear if 0 were in the I-register. However, the negative number *will* be considered by the $\int_x^y$ algorithm in determining the uncertainty of an approximation. The minimum number that can be considered in determining the uncertainty of an approximation is –6. If the I-register contains a number less than –6, the approximation will be performed as if –6 were in the I-register.

Now, slide the PRGM-RUN switch back to ▮▮ RUN . To calculate *Si*(2), key the limits of integration into the X- and Y-registers, then press $\boxed{\text{A}}$ to execute the program.

Keystrokes	Display	
$\boxed{\text{f}}$ $\boxed{\text{SCI}}$ 3	**0.000 00**	Specify $\boxed{\text{SCI}}$ display format. Executing the subsequent program (by pressing $\boxed{\text{A}}$) will change the number of digits specified from 3 to 9. (Display shown assumes no results remain from preceding example.)
0 $\boxed{\text{ENTER}\uparrow}$	**0.000 00**	Key lower limit into Y-register.
2	**2.**	Key upper limit into X-register.
$\boxed{\text{g}}$ $\boxed{\text{RAD}}$	**2.000 00**	Ensure that trigonometric mode is set to radians. (This step is not necessary if you have not switched your calculator off nor reset the trigonometric mode since you last set it to radians.)
$\boxed{\text{A}}$	**1.605412 00**	*Si*(2) calculated with maximum accuracy.
$\boxed{x \gtrless y}$	**6.000000–10**	Uncertainty of approximation.
$\boxed{x \gtrless y}$	**1.605412 00**	Return approximation to display.
$\boxed{\text{h}}$ $\boxed{\text{MANT}}$	**1605412977**	All 10 digits of approximation.

Since the most significant digit of the uncertainty occurs in the tenth decimal place, the uncertainty indicates that the estimate is correct to at least nine decimal places. Indeed, the estimate agrees to all nine decimal places with the value given for *Si* (2) in tables of mathematical functions.

Obtaining the Current Approximation to an Integral

Pressing **R/S** while your HP-34C is calculating an integral halts the calculation, just as it halts the execution of a running program. When you do so, the calculator stops at the current program line in the subroutine you wrote for evaluating the function, and displays the result of executing the preceding program line. Note that after you halt the calculation, the current approximation to the integral is *not* the number in the X-register nor the number in any other stack register. Just as with any program, pressing **R/S** again starts the calculation from the program line at which it was stopped.

When the calculation of an integral is requiring more time than you care to wait, you may want to stop and display the current approximation. You can obtain the current approximation, but not its uncertainty. The $\int_x^y$ algorithm updates the current approximation and stores it in the LAST X register after evaluating the function at each new sample point. To obtain the current approximation, therefore, simply halt the calculator, single-step if necessary through your function subroutine until the calculator has finished evaluating the function and updating the current approximation, then recall the contents of the LAST X register.

Note that while the calculator is updating the current approximation, the display does not flash as it usually does while the calculator is executing your function subroutine. Therefore, you might avoid having to single-step through your subroutine by halting the calculator at a moment when the display is blank.

In summary, to obtain the current approximation to an integral, follow the steps below.

1. Press **R/S** to halt the calculator, preferably while the display is blank.
2. When the calculator halts with a number in the display, slide the PRGM-RUN switch to PRGM.
   a. If the display shows the program line containing the label of your function subroutine, slide the PRGM-RUN switch back to RUN and proceed with step 3.

b. If you didn't press [R/S] at a moment when the display was blank, the display will now show some other program line within your subroutine. Slide the PRGM-RUN switch back to ▮▯RUN and press [h] [SST] repeatedly until the display shows ***25 12*** at the right (or ***000–*** at the left*) while the [SST] key is held down; then release the key and wait for the calculator to halt with a number in the display.

3. Press [h] [LST x]. The current approximation will appear in the display. If you want to continue calculating the final approximation, press [CLX] [+] [R/S]. This refills the stack with the current *x* value and restarts the calculator.

For example, let's calculate the integral *Si* (2) again and obtain the current approximation after a minute or two.

Keystrokes	**Display**	
[g] [R↓] [g] [R↓]	***2.000000 00***	Roll down stack until upper limit appears in X-register.
[A]	***(flashing)***	Start calculation of integral.

After a minute or two, halt the calculator and check the current approximation:

Keystrokes	**Display**	
[R/S]	***6.771087–01***	Halt the calculator by pressing [R/S] while the display is blank. (Of course, the particular number in *your* calculator's display depends on the moment you pressed [R/S].)

* This will occur only when you have not included a [RTN] instruction at the end of your subroutine.

Now slide the PRGM-RUN switch to PRGM to verify that the calculator has stopped at the label of your subroutine.

Display	
001 – 25, 13, 2	Label 2.

Since the calculator stopped at the label of your subroutine, you can recall the current approximation from the LAST X register after sliding the PRGM-RUN switch back to RUN .

Keystrokes	Display	
$\boxed{\text{h}}$ $\boxed{\text{LST x}}$	*1.605412 00*	Current approximation to integral. (Again, the particular number in *your* calculator's display depends on the moment you pressed $\boxed{\text{R/S}}$.)

To continue with the calculation and obtain the final approximation:

Keystrokes	Display	
$\boxed{\text{CLx}}$ $\boxed{+}$	*6.771087–01*	Return current x value to X-register.
$\boxed{\text{R/S}}$	*1.605412 00*	Final approximation to integral.

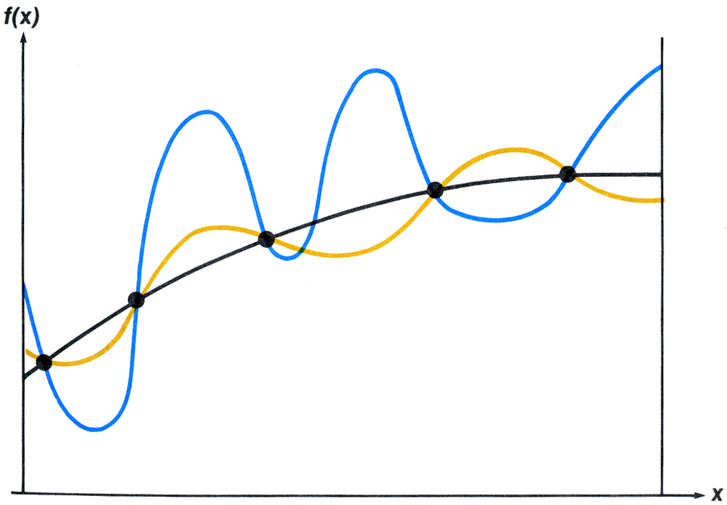
Considerations That Could Cause Incorrect Results

Although the $\boxed{\int_x^y}$ algorithm in your HP-34C is one of the best available, in certain situations it—like nearly all algorithms for numerical integration—might give you an incorrect answer. *The possibility of this occurring is extremely remote*. The $\boxed{\int_x^y}$ 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.

Let's take a more detailed look at the operation of the $\boxed{\int_x^y}$ algorithm to see how it might calculate an incorrect answer. This will enable us to identify the general characteristics of functions that could cause problems. Finally, we'll see how you can verify the accuracy of an approximation if you should ever want to.

As we discussed on page 236, the $\boxed{\int_x^y}$ algorithm samples the function $f(x)$ at various values of x within the interval of integration. By calculating a weighted average of the function's values at the sample points, the algorithm approximates the integral of $f(x)$.

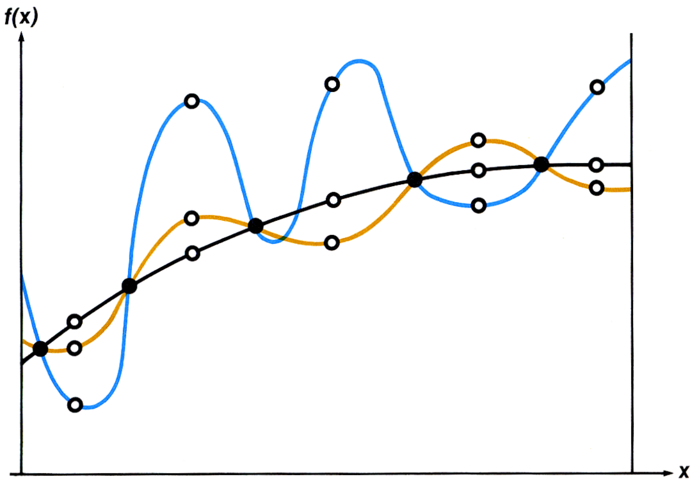
Unfortunately, since all that the algorithm knows about $f(x)$ are its values at the sample points, it cannot distinguish between $f(x)$ and any other function that agrees with $f(x)$ at all the sample points. This situation is depicted in the illustration below, which shows (over a portion of the interval of integration) three of the infinitely many functions whose graphs include the finitely many sample points.



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 in black and gold 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 in blue is quite different from those of the others, so the current approximation will be rather inaccurate if $f(x)$ is this function.

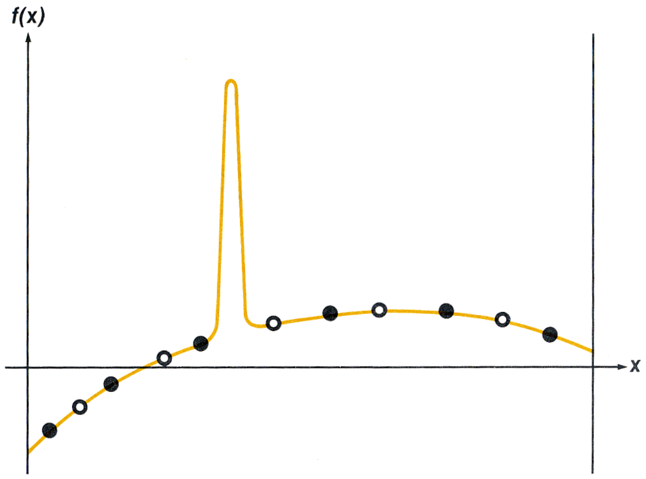
Suppose that the approximation using this number of sample points differs from previous approximations by less than the uncertainty, which was derived from the number of digits specified in the display format. The $\boxed{\int_y^x}$ algorithm will then terminate, returning the current approximation as the best approximation to the integral *given the uncertainty you have implicitly agreed to tolerate*. Thus, for certain functions—such as the function shown in blue—the calculator can give you a rather inaccurate approximation *because it samples the function at only a finite number of points*. This situation represents the extreme case of the trade-off we mentioned earlier (page 241) between accuracy and calculation time: because you don't want to wait an infinitely long time (to sample the function at an infinite number of points), you can't be *absolutely* confident that the calculator's approximation is as accurate as its uncertainty indicates.

Suppose, in contrast to the situation above, that the derived uncertainty in the approximation is so small (because you have specified sufficiently many digits in the display format) that the approximation to the integral using this number of sample points is not sufficiently accurate. The algorithm will then sample $f(x)$ at additional sample points. This situation is depicted in the next illustration, which shows the same three possible functions whose graphs include the first set of sample points.



Although all three functions shown in this illustration have identical values at the smaller number of sample points, the function shown in blue has very different values at the new sample points. When the algorithm processes these new function values, it will find that the disparity between the current approximation and the previous ones is much larger than the acceptable uncertainty. Consequently, the algorithm will continue evaluating the function at more and more sample points until successive approximations agree sufficiently closely. In this case, the calculator can give you an accurate approximation because, in saying that you would accept only a relatively small uncertainty, you agreed to wait as long as necessary.

Practically speaking, however, you wouldn't want to wait forever for an answer. (You probably wouldn't need it then!) By imposing this restriction on the algorithm, you must accept that the function cannot be evaluated at infinitely many sample points and that consequently a sharp and narrow "spike" in the function can be overlooked by the algorithm. This situation is depicted in the next illustration, which shows a function that is smooth except for a prominent spike.



Despite a relatively high density of sample points, none of the sample points happens to discover the spike in the function. Since the approximations after successive iterations agree quite closely, the algorithm would terminate with an approximation that is significantly incorrect because the spike remains undetected by the algorithm.

Why is the spike not detected? Because it is so unlike the mild behavior of the function elsewhere throughout the interval of integration. Except for the spike, the function is smooth throughout the interval shown in the illustration. (Actually, if you viewed the graph of these functions over the entire interval of integration, they might not appear smooth but instead exhibit rapid fluctuations. The illustrations show an expanded view of a small portion of the interval of integration, so that characteristic rapid variations in the functions appear to be smooth.) By sampling the function with sample points of sufficient density, the algorithm comes to know the general behavior of the function. If the spike were not so unlike the rest of the function, either it or similar variations would be detected by the algorithm at some iteration. 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 we're evaluating this integral numerically, we might think (naively in this case, as we will see later) that we should represent the upper limit of integration by 10^{99}, which is virtually the largest number you can key into the calculator. Let's try it and see what happens.

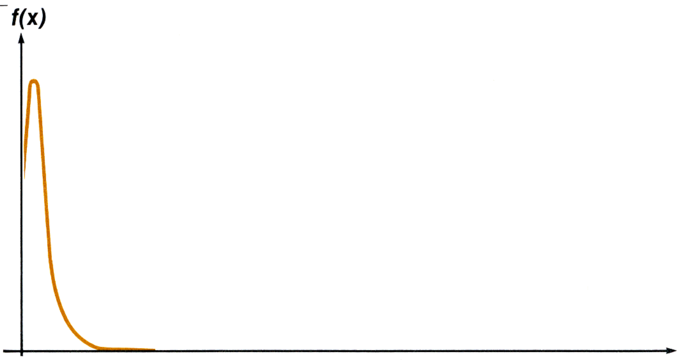
Slide the PRGM-RUN switch to PRGM and key in a subroutine that evaluates the function $f(x) = xe^{-x}$.

Keystrokes	Display
h LBL 1	*001– 25, 13, 1*
CHS	*002– 32*
g e^x	*003– 15 1*
×	*004– 61*
h RTN	*005– 25 12*

Now slide the PRGM-RUN switch back to RUN, set the display format to SCI 3, and key the limits of integration into the X- and Y-registers.

Keystrokes	Display	
f SCI 3	*0.000 00*	Set display format to SCI 3. (Display shown assumes no results remain from preceding example.)
0 ENTER↑	*0.000 00*	Key lower limit into Y-register.
EEX 99	*1. 99*	Key upper limit into X-register.
f $\int_y^x$ 1	*0.000 00*	Approximation of integral.

The answer returned by the calculator is clearly incorrect, since the actual integral of $f(x) = xe^{-x}$ from 0 to ∞ is exactly 1. But the problem is *not* that we represented ∞ by 10^{99}, since the actual integral of this function from 0 to 10^{99} is very close to 1. The reason we got an incorrect answer becomes apparent if we look at the graph of $f(x)$ over the interval of integration:



The graph is a spike very close to the origin. (Actually, to illustrate $f(x)$ we have considerably exaggerated the width of the spike. Shown in actual scale over the interval of integration, the spike would be indistinguishable from the vertical axis of the graph.) 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 $\boxed{\text{SCI}}$ 9, none of the additional sample points would discover the spike when this particular function is integrated over this particular interval. We'll mention a better solution after we briefly describe the general nature of functions that could cause problems.

We have seen how the $\boxed{\int_y^x}$ algorithm can give you an incorrect answer when $f(x)$ has a wiggle somewhere that is very uncharacteristic of the behavior of the function elsewhere. Fortunately, functions exhibiting such aberrations are unusual enough that you are unlikely to have to integrate one unknowingly.

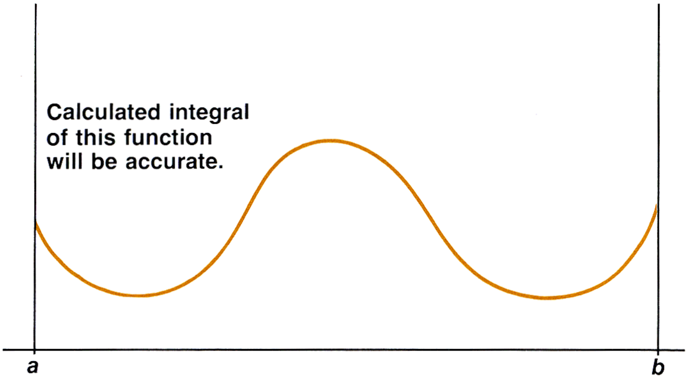
Functions that could lead to incorrect results can be identified most precisely by describing them from the mathematical viewpoint of complex analysis.* But in more simple terms, such a function can be identified by how rapidly it and its low-order derivatives vary across the interval of integration. 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 $\boxed{\int_y^x}$ algorithm terminate, and the less reliable will the resulting approximation be.

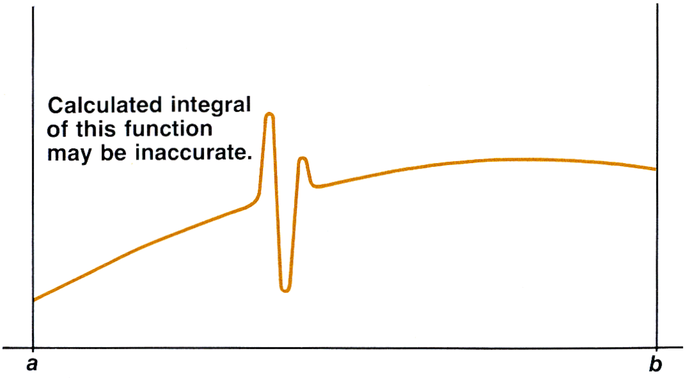
Note that the rapidity of variation in the function (or its low-order derivatives) must be determined with respect to the width of the interval of integration. With a given number of sample points, a function $f(x)$ that has three "wiggles" can be better characterized by its samples when these variations 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 next two illustrations.) Considering the variations or wiggles 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 algorithm will terminate, 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'll know whether the function has any quick wiggles relative to the interval of integration. If you're not familiar with the function, and you have reason to 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 have reason to suspect its validity, there's a very simple procedure you can use 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 approximations over the subintervals.

* The approximations computed by the HP-34C will converge rapidly to the correct answer provided the integrand $f(z)$, regarded as an analytic function of the complex variable z, has no singularities on nor too near the interval of integration, and has an average value on that interval not drastically smaller than its magnitude near that interval.





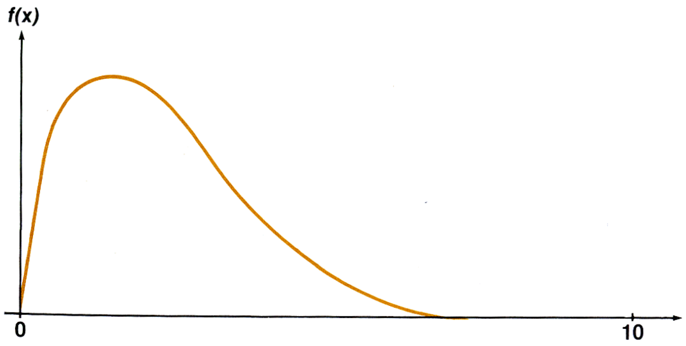
Considerations That Prolong Calculation Time

In the example on page 257, we saw that 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, we would get the correct answer; but it would take a very long time if the interval were still too wide.

For certain integrals, such as the one in that example, calculating the integral may be unduly prolonged because the width of the interval of integration is too large relative to certain features of the function being integrated. Let's 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, we can evaluate the integral by replacing ∞, the upper limit of integration, by a number not so large as 10^{99}, say 10^3.

Keystrokes	Display	
0 ENTER↑	0.000 00	Key lower limit into Y-register.
EEX 3	1. 03	Key upper limit into X-register.
f ∫ʸₓ 1	1.000 00	Approximation to integral.
x≷y	1.824 -04	Uncertainty of approximation.

This is the correct answer, but it took a very long time. To understand why, compare the graph of the function over the interval of integration, which looks about identical to that shown on page 258, to the graph of the function between $x = 0$ and $x = 10$.



By comparing the two graphs, you can see that the function is ''interesting'' only at small values of x. At greater values of x, the function is ''uninteresting,'' since it decreases smoothly and gradually in a very predictable manner.

As we discussed earlier, the $\int_y^x$ algorithm will sample the function with higher densities of sample points until the disparity between successive approximations becomes sufficiently small. In other words, the algorithm samples the function at increasing numbers of sample points until it has sufficient information about the function to provide an approximation that changes insignificantly when further samples are considered.

If the interval of integration were (0, 10) so that the algorithm needed to sample the function only at values where it was interesting but relatively smooth, the sample points after the first few iterations would contribute no new information about the behavior of the function. Therefore, only a few iterations would be necessary before the disparity between successive approximations became sufficiently small that the algorithm could terminate with an approximation of a given accuracy.

On the other hand, if the interval of integration were more like the one shown in the graph on page 261, most of the sample points would capture the function in the region where its slope is not varying much. The few sample points at small values of x would find that values of the function changed appreciably from one iteration to the next. Consequently, the function would have to be evaluated at additional sample points before the disparity between successive approximations would become sufficiently small.

In order for the integral to be approximated with the same accuracy over the larger interval as over the smaller interval, the density of the sample points must be the same in the region where the function is interesting. 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. We will discuss two techniques of doing so: subdividing the interval of integration, and transformation of variables.

Subdividing the Interval of Integration

In regions where the slope of $f(x)$ is varying appreciably, a high density of sample points is necessary to provide an approximation that changes insignificantly from one iteration to the next. However, in regions where the slope of the function stays nearly constant, a high density of sample points is not necessary. This is because evaluating the function at additional sample points would not yield much new information about the function, so it would not dramatically affect the disparity between successive approximations. Consequently, in such regions an approximation of comparable accuracy could be achieved with substantially fewer sample points; so much of the time spent evaluating the function in these regions is wasted. When integrating such functions, you can save time by using the following procedure:

1. Divide the interval of integration into subintervals over which the function is interesting and subintervals over which the function is uninteresting.
2. Over the subintervals where the function is interesting, calculate the integral in the display format corresponding to the accuracy you would like overall.
3. Over the subintervals where the function either is not interesting or contributes negligibly to the integral, calculate the integral with less accuracy, that is, in a display format specifying fewer digits.
4. To get the integral over the entire interval of integration, add together the approximations and their uncertainties from the integrals calculated over each subinterval. You can do this easily using the $\boxed{\Sigma+}$ key.

Before subdividing the interval of integration, check whether the calculator underflows when evaluating the function around the upper (or lower) limit of integration.* Since there is no point in evaluating the function at values of x for which the calculator underflows, in some cases the upper limit of integration can be reduced, saving considerable calculation time.

Remember that once you have keyed in the subroutine that evaluates $f(x)$, you can calculate $f(x)$ for any value of x by keying that value into the X-register and pressing ENTER↑ ENTER↑ ENTER↑ GSB followed by the label of the subroutine.

If the calculator underflows at the upper limit of integration, try smaller numbers until you get closer to the point where the calculator no longer underflows.

Keystrokes	Display	
EEX 3	1. 03	Key upper limit into X-register.
ENTER↑ ENTER↑ ENTER↑	1.000 03	Fill the stack with x.
GSB 1	0.000 00	Calculator underflows at the upper limit.
300 ENTER↑ ENTER↑ ENTER↑	3.000 02	Try a smaller value of x.
GSB 1	0.000 00	Calculator still underflows.
200 ENTER↑ ENTER↑ ENTER↑	2.000 02	Try a smaller value of x.
GSB 1	2.768 -85	Calculator does not underflow at $x = 200$; try a number between 200 and 250.
225 ENTER↑ ENTER↑ ENTER↑	2.250 02	
GSB 1	4.324 -96	Calculator is close to underflow.

At this point, you can use SOLVE to pinpoint the smallest value of x at which the calculator underflows.

* Remember that when the calculation of any quantity would result in a number less than 10^{-99}, the result is replaced by zero. This condition is known as underflow.

Keystrokes	Display		
g R↓	**2.250**	**02**	Roll down stack until the last value tried is in the X- and Y-registers.
f SOLVE 1	**2.280**	**02**	The minimum value of x at which the calculator underflows is about 228.

We have now determined that we need integrate only from 0 to 228. Since the function is interesting only for values of x less than 10, let's divide the interval of integration there. The problem has now become:

$$\int_0^{1000} xe^{-x}\,dx \approx \int_0^{228} xe^{-x}\,dx = \int_0^{10} xe^{-x}\,dx + \int_{10}^{228} xe^{-x}\,dx.$$

Keystrokes	Display		
0 ENTER↑	**0.000**	**00**	Key in lower limit of integration over first subinterval.
10	**10.**		Key in upper limit of integration over first subinterval.
f ∫ʸₓ 1	**9.995**	**-01**	Integral over (0, 10) calculated in SCI 3.
f CLEAR Σ	**9.995**	**-01**	Clear statistical storage registers.
f Σ+	**1.000**	**00**	Sum approximation and its uncertainty in registers R_1 and R_3.
x⇄y	**1.841**	**-04**	Uncertainty of approximation.
g R↓ g R↓	**1.000**	**01**	Roll down stack until upper limit of first integral appears in X-register.

Keystrokes	Display		
228	***228.***		Key upper limit of second integral into X-register. Upper limit of first integral is lifted into Y-register, becoming lower limit of second integral.
$\boxed{\text{f}}$ $\boxed{\text{SCI}}$ 0	***2.***	***02***	Specify $\boxed{\text{SCI}}$ 0 display format for a quick calculation over (10, 228). If the uncertainty of the approximation turns out not to be accurate enough, we can repeat the approximation in a display format specifying more digits.
$\boxed{\text{f}}$ $\boxed{\int_y^x}$ 1	***5.***	***–04***	Integral over (10, 228) calculated in $\boxed{\text{SCI}}$ 0.
$\boxed{\text{f}}$ $\boxed{\text{SCI}}$ 3	***5.328***	***–04***	Change display format back to $\boxed{\text{SCI}}$ 3.
$\boxed{x \gtrless y}$	***7.568***	***–05***	Check uncertainty of approximation. Since it is less than the uncertainty of the approximation over the first subinterval, $\boxed{\text{SCI}}$ 0 yielded an approximation of sufficient accuracy.
$\boxed{x \gtrless y}$	***5.328***	***–04***	Return approximation and its uncertainty to the X- and Y-registers, respectively, before summing them in statistical storage registers.
$\boxed{\text{f}}$ $\boxed{\Sigma+}$	***2.000***	***00***	Sum approximation and its uncertainty.
$\boxed{\text{RCL}}$ $\boxed{\text{f}}$ $\boxed{\Sigma+}$	***1.000***	***00***	Integral over total interval (0, 228).
$\boxed{x \gtrless y}$	***2.598***	***–04***	Uncertainty of integral.

Calculating the integral over the two subintervals took only a fraction of the time to calculate the integral over (0,228); and the combined uncertainty of the total approximation is not appreciably larger than the uncertainty of the single approximation over the entire interval.

Transformation of Variables

In many problems where the function changes very slowly over most of a very wide interval of integration, a suitable transformation of variables may decrease the time required to calculate the integral.

For example, consider again the integral

$$\int_0^\infty xe^{-x}\,dx.$$

Let $u = e^{-x}$.

Then $x = -\ln u$

and $dx = -\dfrac{du}{u}$.

Substituting,

$$\int_0^\infty xe^{-x}\,dx = \int_{e^{-0}}^{e^{-\infty}} (-\ln u)\,(u)\left(-\frac{du}{u}\right)$$

$$= \int_1^0 \ln u\,du.$$

Slide the PRGM-RUN switch to PRGM ▮▮▮ and key in a subroutine that evaluates the function $f(u) = \ln u$.

Keystrokes	Display
h LBL 3	001– 25, 13, 3
f LN	002– 14 1
h RTN	003– 25 12

Slide the PRGM-RUN switch back to ▮▯ RUN and key in the limits of integration, then press f ∫ₓʸ 3 to calculate the integral.

Keystrokes	Display	
1 ENTER↑	1.000 00	Key in lower limit of integration.
0	0.	Key in upper limit of integration.
f ∫ₓʸ 3	9.998 -01	Approximation to equivalent integral.
x⇄y	2.130 -04	Uncertainty of approximation.

Considering the uncertainty of this approximation, it agrees with the value calculated above for the original integral. Yet, it required only a fraction of the calculation time.

Appendix C

Service and Maintenance

Your Hewlett-Packard Calculator

Your calculator is another example of the award-winning design, superior quality, and attention to detail in engineering and construction that have marked Hewlett-Packard electronic instruments for more than 30 years. Each Hewlett-Packard calculator is precision crafted by people who are dedicated to giving you the best possible product at any price.

AC Line Operation

Your calculator contains a rechargeable battery pack consisting of nickel-cadmium batteries. When you receive your calculator, the battery pack inside may be discharged, but you can operate the calculator immediately by using the ac adapter/recharger.

> **Note:** Do not attempt to operate the calculator from an ac line with the battery pack removed.

The procedure for using the ac adapter/recharger is as follows:

1. You need not turn the calculator off.
2. Insert the ac adapter/recharger plug into the connector on the top of the calculator, with the snap release tab on the plug facing toward the right side of the calculator.
3. Insert the power plug into a live ac power outlet.

> **Note:** It is normal for the ac adapter/recharger (and the battery pack) to be warm to the touch when plugged into an ac outlet.

CAUTION

The use of a charger other than the HP recharger supplied with the calculator may result in damage to your calculator.

Use only the "B" suffix version ac adapter/recharger shipped with your calculator (see product number on recharger). Earlier "A" suffix version rechargers will not damage your calculator, but may clear continuous memory when plugged in.

Battery Operation

To operate the calculator from battery power alone, simply disconnect the recharger plug from the calculator by grasping the plug between your thumb and forefinger, squeezing to depress the snap release tab, and pulling gently. (Even when not connected to the calculator, the ac adapter/recharger may be left plugged into the ac outlet.)

Using the calculator on battery power gives the calculator full portability, allowing you to carry it nearly anywhere. A fully charged battery pack typically provides 3 hours of continuous operation. By turning the power off when the calculator is not in use, the charge on the battery pack should easily last throughout a normal working day.

Low Power

When you are operating from battery power and the batteries get low, a raised decimal is turned on at the far left of the display to warn you that you have between 1 minute and 25 minutes of operating time left.

˙***1.23***

If the display contains the low power indication, the minus sign looks like an incomplete divide sign.

∸***1.23***

To return to full power either connect the ac adapter/recharger to the calculator as described under AC Line Operation, or substitute a fully charged battery pack for the one in the calculator.

Battery Charging

The rechargeable batteries in the battery pack are charged while you operate the calculator from the ac adapter/recharger. Batteries will charge with the calculator on or off, provided batteries are in place and recharger is connected. Normal charging times between the fully discharged state and the fully charged state are (depending on ac line voltage value):

Calculator off:	5 to 9 hours
Calculator on:	17 hours

Shorter charging periods will reduce the operating time you can expect from a single battery charge. Whether the calculator is off or on, the calculator battery pack is never in danger of becoming overcharged.

Note: The ac adapter/recharger is a sealed unit and is not repairable. Return it to Hewlett-Packard if service is required.

Using Continuous Memory

When you turn your calculator off, the following information is retained:

- All programs that are loaded into the calculator.
- Contents of the storage registers.
- Display status (FIX, SCI, or ENG, and number of displayed digits).

Regardless of where you stopped in a program, the calculator returns to line 000 (top of program memory) when you turn it on again.

Numbers in the stack, LAST X, and trigonometric mode status (DEG, RAD, or GRAD) are not preserved when you turn the calculator off. Also, all flags and pending subroutines are cleared.

Continuous memory requires that the batteries be kept in the calculator. If the low power indicator appears in the display, turn your calculator off immediately and connect it to an ac outlet or insert a new battery pack. If you allow the battery to discharge completely, the information in continuous memory will be lost.

If you drop or traumatize your calculator, or if power to the continuous memory is interrupted, whether the calculator is off or on, the contents of program memory and the data storage registers may be lost. If this occurs, when the calculator is next turned on with power available, ***Pr Error*** (power failure) will appear in the display. (Pressing any key will clear this and all other error signals.)

Battery Pack Replacement

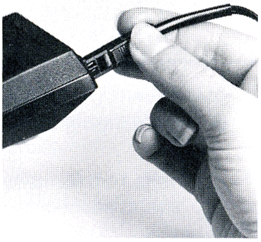
If it becomes necessary to replace the battery pack, use only another Hewlett-Packard battery pack like the one shipped with your calculator. Continuous memory requires that batteries be replaced as quickly as possible. Normally you have a minimum of 5 seconds to change the batteries. Leaving batteries out of the calculator for extended periods will result in loss of information in continuous memory.

CAUTION

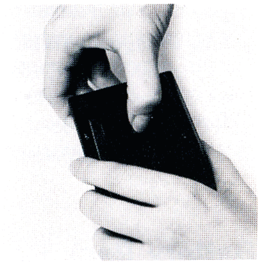
Use of any batteries other than the Hewlett-Packard battery pack may result in damage to your calculator.

To replace the battery pack use the following procedure:

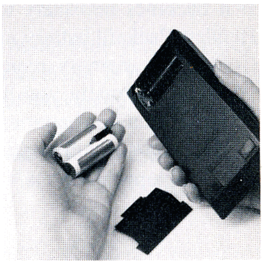
1. Set calculator ON-OFF switch to OFF and disconnect the ac adapter/recharger from the calculator.



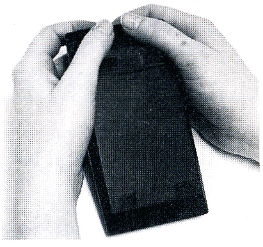
2. Press down on the short ridges of the battery door, close to the edge, until the door release snaps open. Slide the door open.



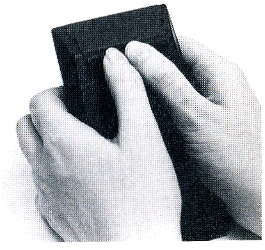
3. When door is removed, turn calculator over and gently shake, allowing the battery pack to fall into the palm of your hand.



4. Place the new battery pack into the calculator. Your calculator will turn on only if the battery pack is inserted correctly.



5. Insert battery door and slide door back into place.



6. Turn calculator over and turn power on to assure proper battery installation. If the display does not light, make sure the battery pack is correctly placed in calculator.

Battery Care

When not being used, the batteries in your calculator have a self-discharge rate of approximately 1 percent of available charge per day. After 30 days, a battery pack might have only 50 to 75 percent of its charge remaining, and the calculator might not even turn on. If a calculator fails to turn on, you should substitute a charged battery pack, if available, for the one in the calculator, or plug in the ac adapter/recharger. The discharged battery pack should be charged for at least 12 hours.

If a battery pack will not hold a charge and seems to discharge very quickly in use, it may be defective. If the one-year warranty on the battery pack has not expired, return the defective pack to Hewlett-Packard according to the shipping instructions. (If you are in doubt about the cause of the problem, return the complete calculator along with its battery pack and ac adapter/recharger.) If the battery pack is out of warranty, see your nearest dealer to order a replacement.

WARNING

Do not attempt to incinerate or mutilate the battery pack—the pack may burst or release toxic materials.

Do not connect together or otherwise short-circuit the battery terminals—the pack may melt or cause serious burns.

Temperature Range

Temperature ranges for the calculator are:

Operating	0° to 45°C	32° to 113°F
Charging	15° to 40°C	59° to 104°F
Storage	−40° to 55°C	−40° to 131°F

Service

Blank Display

If the display blanks out, turn the calculator off, then on. If the display remains blank, check the following:

1. If the ac adapter/recharger is attached to the calculator, make sure it is plugged into an ac outlet.
2. Examine the battery pack to see if the contacts are dirty.
3. Substitute a fully charged battery pack, if available, for the one that was in the calculator.
4. If the display is still blank, try operating the calculator using the ac adapter/recharger (with the batteries in the calculator).
5. If, after step 4, the display is still blank, service is required. (Refer to Limited One-Year Warranty.)

Limited One-Year Warranty

What We Will Do

The HP-34C and its accessories are warranted by Hewlett-Packard against defects in materials and workmanship for one year from date of original purchase. If you sell your calculator 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 that you return the product, shipping prepaid, to a Hewlett-Packard repair center.

How to Obtain Repair Service

Hewlett-Packard maintains repair centers in most major countries throughout the world. You may have your calculator repaired at a Hewlett-Packard repair center anytime it needs service, whether the unit is under warranty or not. There is a charge for repairs after the one-year warranty period. Please refer to the Shipping Instructions in this handbook.

Hewlett-Packard calculators are normally repaired and reshipped within five (5) working days of receipt at any repair center. This is an average time and could possibly vary depending upon time of year and work load at the repair center.

The Hewlett-Packard United States Repair Center for handheld and portable printing calculators is located at Corvallis, Oregon. The mailing address is:

Hewlett-Packard
Corvallis Division • Service Department
1000 N.E. Circle Boulevard/P.O. Box 999
Corvallis, Oregon 97330

What Is Not Covered

This warranty does not apply if the product has been damaged by accident or misuse, or as a result of service or modification by other than an authorized Hewlett-Packard repair center.

No other expressed warranty is given. The repair or replacement of a product is your exclusive remedy. **ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY.** Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Obligation to Make Changes

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.

Warranty Information Toll-Free Number

If you have any questions concerning this warranty, please call 800/648-4711. (In Nevada call 800/992-5710.)

Shipping Instructions

The calculator should be returned, along with completed Service Card, in its shipping case (or other protective package) to avoid in-transit damage. Such damage is not covered by warranty and Hewlett-Packard suggests that the customer insure shipments to the repair center. A calculator returned for repair should include the ac adapter/recharger and the battery pack. Send these items to the address shown on the Service Card. Remember to include a sales slip or other proof of purchase with your unit.

Whether the unit is under warranty or not, it is your responsibility to pay shipping charges for delivery to the Hewlett-Packard repair center.

After warranty repairs are completed, the repair center returns the unit with postage prepaid. On out-of-warranty repairs, the unit is returned C.O.D. (covering shipping costs and the service charge).

Programming and Applications Assistance

Should you need technical assistance concerning programming, calculator applications, etc., call Hewlett-Packard Customer Support at 503/757-2000. This is not a toll-free number, and we regret that we cannot accept collect calls. As an alternative, you may write to:

Hewlett-Packard
Corvallis Division Customer Support
1000 N.E. Circle Boulevard
Corvallis, OR 97330

A great number of our users submit program applications or unique program key sequences to share with other HP owners. Hewlett-Packard will only consider using ideas given freely to us. Since it is the policy of Hewlett-Packard not to accept suggestions given in confidence, the following statement must be included with your submittal:

''I am voluntarily submitting this information to Hewlett-Packard Company. The information is not confidential and Hewlett-Packard may do whatever it wishes with the information without obligation to me or anyone else.''

Further Information

Service contracts are not available. Calculator circuitry and design are proprietary to Hewlett-Packard, and service manuals are not available to customers. Should problems arise regarding repairs, please contact your nearest Hewlett-Packard repair center. The address for the United States Repair Center for handheld and portable printing calculators is:

Hewlett-Packard Company
Corvallis Division • Service Department
1000 N.E. Circle Boulevard/P.O. Box 999
Corvallis, Oregon 97330

Note: Not all Hewlett-Packard repair centers offer service for all models of HP calculators. However, you can be sure that service may be obtained in the country where you bought your calculator.

If you happen to be outside of the country where you bought your calculator, you can contact the local Hewlett-Packard repair center to see if service capability is available for your model. If service is unavailable, ship your calculator to the above address. A list of repair centers for other countries may be obtained by writing to the above address.

All shipping and reimportation arrangements are your responsibility.

Appendix D

Error Conditions

If you attempt a calculation containing an improper operation—say division by zero—the display will show ***Error*** and a number. To clear an error message, press any key.

The following operations will display ***Error*** plus a number:

Error 0: **Improper Mathematical Operation**

Illegal argument to math routine;
$\boxed{\div}$, where $x = 0$.
$\boxed{y^x}$, where $y = 0$ and $x \leqslant 0$, or $y < 0$ and x is non-integer.
$\boxed{\sqrt{x}}$, where $x < 0$.
$\boxed{1/x}$, where $x = 0$.
$\boxed{\text{LOG}}$, where $x \leqslant 0$.
$\boxed{\text{LN}}$, where $x \leqslant 0$.
$\boxed{\text{SIN}^{-1}}$, where $|x|$ is > 1.
$\boxed{\text{COS}^{-1}}$, where $|x|$ is > 1.
$\boxed{\text{STO}}$ $\boxed{\div}$, where $x = 0$.
$\boxed{\Delta\%}$, where the value in the y-register is 0.

Error 1: **Storage Register Overflow**

Storage register overflow (except $\boxed{\Sigma+}$, $\boxed{\Sigma-}$). Magnitude of number in storage register would be larger than $9.999999999 \times 10^{99}$.

Error 2: **Improper Register Number**

Named storage register currently converted to program memory, or nonexistent storage register.

Error 3: Improper Statistical Operation

$\boxed{\bar{x}}$	$n = 0$
$\boxed{s}$	$n \leqslant 1$
$\boxed{r}$	$n \leqslant 1$
$\boxed{\hat{y}}$	$n \leqslant 1$
$\boxed{\text{L.R.}}$	$n \leqslant 1$

Note: ***Error 3*** is also displayed if division by zero or the square root of a negative number would be required during computation with any of the following formulas:

$$s_x = \sqrt{\frac{M}{n(n-1)}} \qquad s_y = \sqrt{\frac{N}{n(n-1)}} \qquad r = \frac{P}{\sqrt{M \cdot N}}$$

$$A = \frac{P}{M} \qquad B = \frac{M\Sigma y - P\Sigma x}{n . M}$$

(A and B are the values returned by the operation $\boxed{\text{L.R.}}$, where $y = Ax + B$.)

$$\hat{y} = \frac{M\Sigma y + P(n \cdot x - \Sigma x)}{n \cdot M}$$

where:

$$M = n\Sigma x^2 - (\Sigma x)^2$$
$$N = n\Sigma y^2 - (\Sigma y)^2$$
$$P = n\Sigma xy - \Sigma x \Sigma y$$

Error 4: Improper Line Number or Label Call

Line number called for is currently unoccupied, or nonexistent (>210), attempt to load more than 210 lines of program memory, or label called does not exist.

Error 5

Recursive call to $\boxed{\int_y^x}$ or $\boxed{\text{SOLVE}}$, i.e., $\boxed{\int_y^x}$ within a subroutine called by another $\boxed{\int_y^x}$ or $\boxed{\text{SOLVE}}$ within a subroutine called by another $\boxed{\text{SOLVE}}$.

Error 6

SOLVE unable to find a root using given estimates.

Error 7

Illegal label (4-9) used with $\int_y^x$ or SOLVE, or illegal flag name (4-9).

Error 8

Subroutine level too deep.

Error 9

Self-test discovered circuitry problem. Note that program memory, storage register contents, and display setting are not cleared by executing the self-test (STO ENTER↑).

Pr Error

Continuous memory cleared because of power failure.

Appendix E

Stack Lift and LAST X

Your HP-34C calculator has been designed to operate in a natural manner. As you have seen as you worked through this handbook, you are seldom required to think about the operation of the automatic memory stack—you merely work through calculations in the same way you would with a pencil and paper, performing one operation at a time.

There may be occasions, however, particularly as you program the HP-34C, when you wish to know the effect of a particular operation upon the stack. The following explanation should help you.

Digit Entry Termination

Most operations on the calculator, whether executed as instructions in a program or pressed from the keyboard, terminate digit entry. This means that the calculator knows that any digits you key in after any of these operations are part of a new number.

Stack Lift

There are three types of operations on the calculator, depending upon how they affect the stack lift. These are stack-*disabling* operations, stack-*enabling* operations, and *neutral* operations.

Disabling Operations

There are only four stack-disabling operations on the calculator. These operations disable the stack lift, so that a number keyed in after one of these disabling operations writes over the current number in the displayed X-register and the stack does not lift. These special disabling operations are:

[ENTER↑] [CLX] [Σ+] [Σ−]

Enabling Operations

The bulk of the operations on the keyboard, including one- and two-number mathematical functions like [x^2] and [×], are stack enabling operations. These operations enable the stack lift, so that a number keyed in after one of the enabling operations lifts the stack. Note that switching from PRGM mode to RUN mode is an enabling operation.

Neutral Operations

Some operations, like CHS and FIX, are neutral; that is, they do not alter the previous status of the stack lift. Thus, if you disable the stack lift by pressing ENTER↑, then press f FIX *n* and key in a new number, that number will write over the number in the X-register and the stack will not lift. Similarly, if you have previously enabled the stack lift by executing, say x^2, then execute a FIX instruction followed by a digit entry sequence, the stack will lift.

The following operations are neutral on the HP-34C:

FIX	GTO • *nnn*	CLEAR PREFIX
SCI	BST	CLEAR REG
ENG	SST (In RUN mode SST may execute an instruction that does enable the stack.)	CLEAR Σ
DEG		CHS*
RAD		MANT
GRD		R/S
DSP I	MEM	PSE

LAST X

The following operations save x in LAST X:

−	Σ+	e^x	$\sqrt{x}$
+	Σ−	LOG	x^2
×	%	10^x	$1/x$
÷	Δ%	SIN	y^x
→H.MS	$\hat{y}$	SIN^{-1}	→R
→H	$x!$	COS	→P
ABS	FRAC	COS^{-1}	
→R	INT	TAN	
→D	LN	TAN^{-1}	

* CHS is neutral during digit entry of a number from keys, as in 1, 2, 3, CHS to enter −123; or, 123 EEX CHS to enter 123 × 10^{-6}. But otherwise, CHS enables the stack, as you would expect.

HEWLETT hp PACKARD

1000 N.E. Circle Blvd., Corvallis, OR 97330

For additional sales and service information contact your local Hewlett-Packard Sales Office or Call 800/648-4711. (In Nevada call 800/992-5710.)

00034-90001

Printed in U.S.A.