HEWLETT PACKARD

HP-15C

Owner's Handbook

November 1985

00015-90001 Rev. G

Printed in Canada

Introduction

Congratulations! Whether you are new to HP calculators or an experienced user, you will find the HP-15C unmatched in the calculator world. Besides Continuous Memory and low power consumption, the HP-15C state-of-the-art technology provides:

- 448 bytes of program memory (one or two bytes per instruction) and sophisticated programming capability, including conditional and unconditional branching, subroutines, flags, and editing.
- Four advanced mathematics capabilities: complex number calculations, matrix calculations, solving for roots, and numerical integration.
- Direct and indirect storage in up to 67 registers.
- Long-life batteries.

This handbook is written for you, regardless of your level of expertise. The first part, Fundamentals, covers all the basic functions of the HP-15C and how to use them. Each section in the second part, Programming, is broken down into three subsections—The Mechanics, Examples, and Further Information—in order to make it easy for users with varying backgrounds to find the information they need. The third part, Advanced Functions, describes the four advanced mathematics capabilities.*

Before starting these sections, you may want to gain some operating and programming experience on the HP-15C by working through the introductory material, The HP-15C: A Problem Solver, on page 12.

The various appendices describe additional details of calculator operation, as well as warranty and service information. The Function Summary and Index and the Programming Summary and Index at the back of this manual can be used for quick

---

* You certainly do not need to read every page of parts I and II before delving into the HP-15C Advanced Functions if you are already familiar with HP calculators. The use of [SOLVE] and [∫] requires a knowledge of HP-15C programming.

reference to each function key and as a handy page reference to more comprehensive information inside the manual.

Also available from Hewlett-Packard dealers is the *HP-15C Advanced Functions Handbook*, which provides applications and technical descriptions for the root-solving, integration, complex number, and matrix functions.

Contents

The HP-15C: A Problem Solver

The HP-15C Advanced Programmable Scientific Calculator is a powerful problem solver, convenient to carry and easy to hold. Its Continuous Memory retains data and program instructions indefinitely until you choose to reset it. Though sophisticated, it requires no prior programming experience or knowledge of programming languages to use it.

An important new feature of your HP-15C is its extremely low power consumption. This efficiency is responsible for the lightweight, compact design, and eliminates the need for a recharger. Power consumption in the HP-15C is so low that the average battery life in normal use is 6 to 12 months. In addition, the low-power indicator gives you plenty of warning before the calculator stops functioning.

The HP-15C also conserves power by automatically shutting its display off if it is left inactive for a few minutes. But don't worry about losing data—any information contained in the HP-15C is saved by Continuous Memory.

A Quick Look at [ENTER]

Your Hewlett-Packard calculator uses a unique operating logic, represented by the [ENTER] key, that differs from the logic in most other calculators. You will find that using [ENTER] makes nested and complicated calculations easier and faster to work out. Let's get acquainted with how this works.

For example, let's look at the arithmetic functions. First we have to get the numbers into the machine. Is your calculator on? If not, press [ON]. Is the display cleared? To display all zeros, you can press [g] [CLx], that is, press [g], then [←].* To perform arithmetic,

* If you have not used an HP calculator before, you will notice that most keys have three labels. To use the primary function—the one printed in white on top of the key—just press that key. For those printed in gold or blue, press the gold [f] key or the blue [g] key first.

key in the first number, press [ENTER] to separate the first number from the second, then key in the second number and press [+], [−], [×], or [÷]. The result appears immediately after you press any numerical function key.

The display format used in this handbook is [FIX] 4 (the decimal point is "fixed" to show four decimal places) unless otherwise mentioned. If your calculator does not show four decimal places, you may want to press [f] [FIX] 4 to match the displays in the examples.

Manual Solutions

Run through the following two-number calculations. It is not necessary to clear the calculator between problems. If you enter a digit incorrectly, press [←] to undo the mistake, then key in the correct number.

To Compute:	Keystrokes	Display
$9 - 6 = 3$	9 [ENTER] 6 [−]	3.0000
$9 \times 6 = 54$	9 [ENTER] 6 [×]	54.0000
$9 \div 6 = 1.5$	9 [ENTER] 6 [÷]	1.5000
$9^6 = 531{,}441$	9 [ENTER] 6 [y^x]	531,441.0000

Notice that in the four examples:

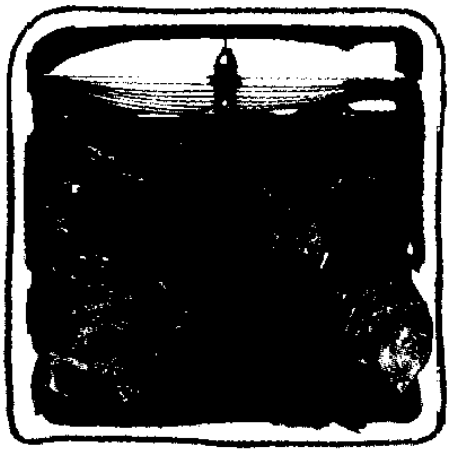
- Both numbers are in the calculator before you press the function key.
- [ENTER] is used only to separate two numbers that are keyed in one after the other.
- Pressing a numeric function key, in this case [−], [×], [÷], or [y^x], executes the function immediately and displays the result.

To see the close relationship between manual and programmed problem solving, let's first calculate the solution to a problem manually, that is, from the keyboard. Then we'll use a program to calculate the solution to the same problem with different data.

The time an object takes to fall to the ground (ignoring air friction) is given by the formula

$$t = \sqrt{\frac{2h}{g}} \ ,$$

where t = time in seconds,
h = height in meters,
g = the acceleration due to gravity, 9.8 m/s^2.



Example: Compute the time taken by a stone falling from the top of the Eiffel Tower (300.51 meters high) to the earth.

Keystrokes	Display	
300.51 [ENTER]	**300.5100**	Enter h.
2 [×]	**601.0200**	Calculates $2h$.
9.8 [÷]	**61.3286**	$(2h)/g$.
[√x]	**7.8313**	Falling time, seconds.

Programmed Solutions

Suppose you wanted to calculate falling times from various heights. The easiest way is to write a program to cover all the constant parts of a calculation and provide for entry of variable data.

Writing the Program. The program is similar to the keystroke sequence you used above. A label is useful to define the beginning of a program, and a return is useful to mark the end of a program. Also, the program must accommodate the entry of new data.

Loading the Program. You can load a program for the above problem by pressing the following keys in sequence. (The display shows information which you can ignore for now, though it will be useful later.)

Keystrokes	Display		
[g] [P/R]	000–		Sets HP-15C to Program mode. (**PRGM** annunciator on.)
[f] CLEAR [PRGM]	000–		Clears program memory. (This step is optional here.)
[f] [LBL] [A]	001–	42,21,11	Label "A" defines the beginning of the program.
2	002–	2	The same keys you pressed to solve the problem manually.
[×]	003–	20	
9	004–	9	
[.]	005–	48	
8	006–	8	
[÷]	007–	10	
[√x]	008–	11	
[g] [RTN]	009–	43 32	"Return" defines the end of the program.
[g] [P/R]	7.8313		Switches to Run mode. (No **PRGM** annunciator.)

Running the Program. Enter the following information to run the program.

Keystrokes	Display	
300.51	300.51	Height of the Eiffel Tower.
[f] [A]	7.8313	Falling time you calculated earlier.
1050 [f] [A]	14.6385	The time (seconds) for a stone to reach the ground after release from a blimp 1050 m high.

With this program loaded, you can quickly calculate the time of descent of an object from different heights. Simply key in the height and press [f][A]. Find the time of descent for objects released from heights of 100 m, 2 m, 275 m, and 2,000 m.

The answers are: 4.5175 s; 0.6389 s; 7.4915 s; and 20.2031 s.

That program was relatively easy. You will see many more aspects and details of programming in part II. For now, turn the page to part I to take an in-depth look at some of the calculator's important operating basics.

Part I

HP-15C Fundamentals

Section 1

Getting Started

Power On and Off

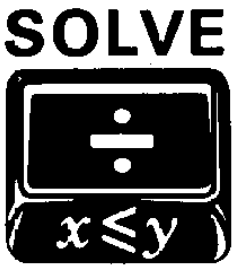
The [ON] key turns the HP-15C on and off.* To conserve power, the calculator automatically turns itself off after a few minutes of inactivity.

Keyboard Operation

Primary and Alternate Functions

Most keys on your HP-15C perform one primary and two alternate, shifted functions. The primary function of any key is indicated by the character(s) on the face of the key. The alternate functions are indicated by the gold characters printed above the key and the blue characters printed on the lower face of the key.

- To select the primary function printed on the face of a key, press only that key. For example: [÷].
- To select the alternate function printed in gold or blue, press the like-colored prefix key ([f] or [g]) followed by the function key. For example: [f] [SOLVE]; [g] [$x \leq y$].



Throughout this handbook, we will observe certain conventions in referring to alternate functions. References to the *function itself* will appear as just the key name in a box, such as "the [MEM] function." References to *the use of the key* will include the prefix key, such as "press [g] [MEM]." References to the four gold functions printed under the bracket labeled "CLEAR" will be preceded by the word "CLEAR," such as "the CLEAR [REG] function," or "press [f] CLEAR [PRGM]."

---

*Note that the [ON] key is lower than the other keys to help prevent its being pressed inadvertently.

Notice that when you press the [f] or [g] prefix key, an **f** or **g** annunciator appears and remains in the display until a function key is pressed to complete the sequence.

Prefix Keys

A prefix key is any key which must precede another key to complete the key sequence for a function. Certain functions require two parts: a prefix key and a digit or other key. For your reference, the prefix keys are:

[CF]	[ENG]	[FIX]	[GSB]	[$\int x$]	[MATRIX]	[SCI]	[STO]
[DIM]	[f]	[g]	[HYP]	[ISG]	[RCL]	[SF]	[TEST]
[DSE]	[F?]	[GTO]	[HYP^{-1}]	[LBL]	[RESULT]	[SOLVE]	[$x \geq$]

If you make a mistake while keying in a prefix for a function, press [f] CLEAR [PREFIX] to cancel the error. The CLEAR [PREFIX] key is also used to show the mantissa of a displayed number, so all 10 digits of the number in the display will appear for a moment after the [PREFIX] key is pressed.

Changing Signs

Pressing [CHS] (*change sign*) will change the sign (positive or negative) of any displayed number. To key in a negative number, press [CHS] after its digits have been keyed in.

Keying in Exponents

[EEX] (*enter exponent*) is used when keying in a number with an exponent. First key in the mantissa, then press [EEX] and key in the exponent.

For a negative exponent press [CHS] after keying in the exponent.* For example, to key in Planck's constant (6.6262×10^{-34} Joule-seconds) and multiply it by 50:

---

* [CHS] may also be pressed after [EEX] and *before* the exponent, with the same result (unlike the mantissa, where digit entry must precede [CHS]).

Keystrokes	Display		
6.6262	**6.6262**		
[EEX]	**6.6262**	**00**	The 00 prompts you to key in the exponent.
3	**6.6262**	**03**	(6.6262×10^{3}).
4	**6.6262**	**34**	(6.6262×10^{34}).
[CHS]	**6.6262**	**-34**	(6.6262×10^{-34}).
[ENTER]	**6.6262**	**-34**	Enters number.
50 [×]	**3.3131**	**-32**	Joule-seconds.

Note: Decimal digits from the mantissa that spill into the exponent field will disappear from the display when you press [EEX], but will be retained internally.

To prevent a misleading display pattern, [EEX] will not operate with a number having more than seven digits to the left of the radix mark (decimal point), nor with a mantissa smaller than 0.000001. To key in such a number, use a form having a greater exponent value (whether positive or negative). For example, $123456789.8 \times 10^{23}$ can be keyed in as $1234567.898 \times 10^{25}$; $0.00000025 \times 10^{-15}$ can be keyed in as 2.5×10^{-22}.

The "CLEAR" Keys

Clearing means to replace a number with zero. The clearing operations in the HP-15C are (the table is continued on the next page):

Clearing Sequence	Effect
[g] [CLx]	Clears display (X-register).
[←]	
In Run mode:	Clears last digit **or** entire display.
In Program mode:	Deletes current instruction.
[f] CLEAR [Σ]	Clears statistics storage registers, display, and the memory stack (described in section 3).

Clearing Sequence	Effect
[f] CLEAR [PRGM]	
In Run mode:	Repositions program memory to line 000.
In Program mode:	Deletes all program memory.
[f] CLEAR [REG]	Clears all data storage registers.
[f] CLEAR [PREFIX] *	Clears any prefix from a partially entered key sequence.

*Also temporarily displays the mantissa.

Display Clearing: [CLx] and [←]

The HP-15C has two types of display clearing operations: [CLx] (*clear X*) and [←] (*back arrow*).

In Run mode:

- [CLx] clears the display to zero.
- [←] deletes only the last digit in the display *if digit entry has not been terminated* by [ENTER] or most other functions. You can then key in a new digit or digits to replace the one(s) deleted. If digit entry *has* been terminated, then [←] acts like [CLx].

Keystrokes	Display	
12345	**12,345**	Digit entry not terminated.
[←]	**1,234**	Clears only the last digit.
9	**12,349**	
[√x]	**111.1261**	Terminates digit entry.
[←]	**0.0000**	Clears all digits to zero.

In Program mode:

- [CLx] is programmable: it is *stored* as a programmed instruction, and will not delete the currently displayed instruction.
- [←] is *not* programmable, so it can be used for program correction. Pressing [←] will delete the entire instruction currently displayed.

Calculations

One-Number Functions

A one-number function performs an operation using only the number in the display. To use any one-number function, press the function key *after* the number has been placed in the display.

Keystrokes	Display
45	45
g LOG	1.6532

Two-Number Functions and ENTER

A two-number function must have two numbers present in the calculator *before* executing the function. +, −, ×, and ÷ are examples of two-number functions.

Terminating Digit Entry. When *keying in* two numbers to perform an operation, the calculator needs a signal that *digit entry is terminated* for the first number. This is done by pressing ENTER to separate the two numbers. If, on the other hand, one of the numbers is already in the calculator as the result of a previous operation, you do not need to use the ENTER key. All functions except the digit entry keys themselves* have the effect of *terminating digit entry*.

Notice that, regardless of the number, a decimal point always appears and a set number of decimal places are displayed when you terminate digit entry (as by pressing ENTER).

Chain Calculations. In the following calculations, notice that:

- The ENTER key is used only for separating the *sequential* entry of two numbers.
- The operator is keyed in *only after* both operands are in the calculator.
- The result of any operation may itself become an operand. Such intermediate results are stored and retrieved on a last-in, first-out basis. New digits keyed in following an operation are treated as a new number.

---

* The digit keys, ·, CHS, EEX, and ←.

Example: Calculate $(9 + 17 - 4) \div 4$.

Keystrokes	Display	
9 [ENTER]	**9.0000**	Digit entry terminated.
17 [+]	**26.0000**	$(9 + 17)$.
4 [-]	**22.0000**	$(9 + 17 - 4)$.
4 [÷]	**5.5000**	$(9 + 17 - 4) \div 4$.

Even more complicated problems are solved in the same manner—using automatic storage and retrieval of intermediate results. It is easiest to work from the inside of parentheses outwards, just as you would with calculations on paper.

Example: Calculate $(6 + 7) \times (9 - 3)$.

Keystrokes	Display	
6 [ENTER]	**6.0000**	First solve for the intermediate result of $(6 + 7)$.
7 [+]	**13.0000**	
9 [ENTER]	**9.0000**	Then solve for the intermediate result of $(9 - 3)$.
3 [-]	**6.0000**	
[×]	**78.0000**	Then multiply the intermediate results together (13 and 6) for the final answer.

Try your hand at the following problems. Each time you press [ENTER] or a function key in a calculation, the preceding number is saved for the next operation.

$$(16 \times 38) - (13 \times 11) = 465.0000$$

$$4 \times (17 - 12) \div (10 - 5) = 4.0000$$

$$23^2 - (13 \times 9) + 1/7 = 412.1429$$

$$\sqrt{[(5.4 \times 0.8) \div (12.5 - 0.7^2)]} = 0.5998$$

Section 2

Numeric Functions

This section discusses the numeric functions of the HP-15C (excluding statistics and advanced functions). The nonnumeric functions are discussed separately (digit entry in section 1, stack manipulation in section 3, and display control in section 5).

The numeric functions of the HP-15C are used in the same way whether executed from the keyboard or in a program. Some of the functions (such as [ABS]) are, in fact, primarily of interest for programming.

Remember that the numeric functions, like all functions except digit entry functions, automatically terminate digit entry. This means a numeric function does not need to be preceded or followed by [ENTER].

Pi

Pressing [g] [π] places the first 10 digits of π into the calculator. [π] does not need to be separated from other numbers by [ENTER].

Number Alteration Functions

The number alteration functions act upon the number in the display (X-register).

Integer Portion. Pressing [g] [INT] replaces the number in the display with the nearest integer of lesser or equal magnitude.

Fractional Portion. Pressing [f] [FRAC] replaces the number in the display with its fractional part (that is, the difference between the number and its integer part).

Rounding. Pressing [g] [RND] rounds all 10 internally held digits of the mantissa of the displayed value to the number of digits specified by the current [FIX], [SCI], or [ENG] display format.

Absolute Value. Pressing [g] [ABS] yields the absolute value of the number in the display.

Keystrokes	Display	
123.4567 [g] [INT]	123.0000	
[g] [LSTx] [CHS] [g] [INT]	−123.0000	Reversing the sign does not alter digits.
[g] [LSTx] [f] [FRAC]	−0.4567	
1.23456789 [CHS] [g] [RND]	−1.2346	
[f] CLEAR [PREFIX]	1234600000	Temporarily displays all digits in the mantissa.
(release)	−1.2346	
[g] [ABS]	1.2346	

One-Number Functions

One-number math functions in the HP-15C operate only upon the number in the display (X-register).

General Functions

Reciprocal. Pressing [1/x] calculates the reciprocal of the number in the display.

Factorial and Gamma. Pressing [f] [x!] calculates the factorial of the displayed value, where x is an integer $0 \leqslant x \leqslant 69$.

You can also use [x!] to calculate the Gamma function, $\Gamma(x)$, used in advanced mathematics and statistics. Pressing [f] [x!] calculates $\Gamma(x+1)$, so you must subtract 1 from your initial operand to get $\Gamma(x)$. For the Gamma function, x is not restricted to nonnegative integers.

Square Root. Pressing [√x] calculates the positive square root of the number in the display.

Squaring. Pressing [g] [x²] calculates the square of the number in the display.

Keystrokes	Display	
25 [1/x]	0.0400	
8 [f] [x!]	40,320.0000	Calculates 8! or Γ(9).
3.9 [√x]	1.9748	
12.3 [g] [x²]	151.2900	

Trigonometric Operations

Trigonometric Modes. The trigonometric functions operate in the trigonometric mode you select. Specifying a trigonometric mode does not convert any number already in the calculator to that mode; it merely tells the calculator what unit of measure (degrees, radians, or grads) to assign a number for a trigonometric function.

Pressing [g] [DEG] sets Degrees mode. No annunciator appears in the display. Degrees are in *decimal*, not minutes-seconds form.

Pressing [g] [RAD] sets Radians mode. The **RAD** annunciator appears in the display. In Complex mode, all functions (except [→P] and [→R]) assume values are in radians, regardless of the trigonometric annunciator displayed.

Pressing [g] [GRD] sets Grads mode. The **GRAD** annunciator appears in the display.

Continuous Memory will maintain the last trigonometric mode selected. At "power up" (initial condition or when Continuous Memory is reset), the calculator is in Degrees mode.

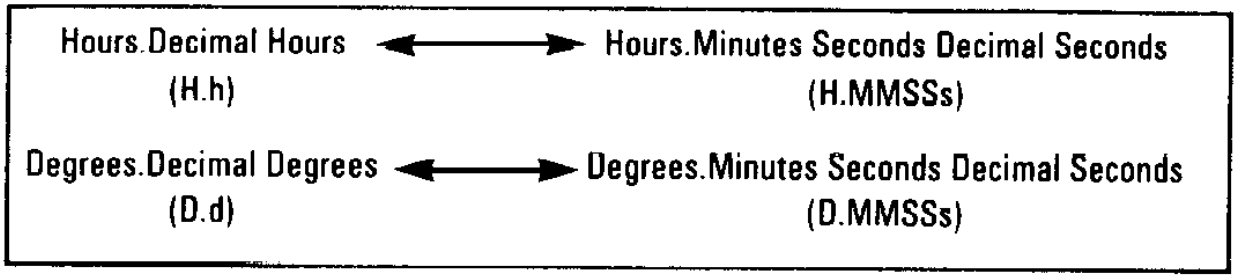
Trigonometric Functions. Given x in the display (X-register):

Pressing	Calculates
[SIN]	sine of x
[g] [SIN^{-1}]	arc sine of x
[COS]	cosine of x
[g] [COS^{-1}]	arc cosine of x
[TAN]	tangent of x
[g] [TAN^{-1}]	arc tangent of x

Before executing a trigonometric function, be sure that the calculator is set to the desired trigonometric mode (Degrees, Radians, or Grads).

Time and Angle Conversions

Numbers representing time (hours) or angles (degrees) can be converted by the HP-15C between a decimal-fraction and a minutes-seconds format:



Hours/Degrees-Minutes-Seconds Conversion. Pressing [f] [→H.MS] converts the number in the display from a decimal hours/degrees format to an hours/degree-minutes-seconds-decimal seconds format.

For example, press [f] [→H.MS] to convert

Press [f] [PREFIX] to display the value to all possible decimal places:

1 1 4 0 4 2 0 0 0 0

to the hundred-thousandths of a second.

Decimal Hours (or Degrees) Conversion. Pressing [g] [→H] converts the number in the display from an hours/degrees-minutes-seconds-decimal seconds format to a decimal hours/degrees format.

Degrees/Radians Conversions

The [→DEG] and [→RAD] functions are used to convert angles to degrees or radians (D.d◄►R.r). The degrees must be expressed as decimal numbers, and not in a minutes-seconds format.

Keystrokes	Display	
40.5 [f] [→RAD]	**0.7069**	Radians.
[g] [→DEG]	**40.5000**	40.5 degrees (decimal fraction).

Logarithmic Functions

Natural Logarithm. Pressing [g] [LN] calculates the natural logarithm of the number in the display; that is, the logarithm to the base e.

Natural Antilogarithm. Pressing [e^x] calculates the natural antilogarithm of the number in the display; that is, raises e to the power of that number.

Common Logarithm. Pressing [g] [LOG] calculates the common logarithm of the number in the display; that is, the logarithm to the base 10.

Common Antilogarithm. Pressing [10^x] calculates the common antilogarithm of the number in the display; that is, raises 10 to the power of that number.

Keystrokes	Display	
45 [g] [LN]	**3.8067**	Natural log of 45.
3.4012 [e^x]	**30.0001**	Natural antilog of 3.4012.
12.4578 [g] [LOG]	**1.0954**	Common log of 12.4578.
3.1354 [10^x]	**1,365.8405**	Common antilog of 3.1354.

Hyperbolic Functions

Given x in the display (X-register):

Pressing	Calculates
[f] [HYP] [SIN]	hyperbolic sine of x
[g] [HYP^{-1}] [SIN]	inverse hyperbolic sine of x
[f] [HYP] [COS]	hyperbolic cosine of x
[g] [HYP^{-1}] [COS]	inverse hyperbolic cosine of x
[f] [HYP] [TAN]	hyperbolic tangent of x
[g] [HYP^{-1}] [TAN]	inverse hyperbolic tangent of x

Two-Number Functions

The HP-15C performs two-number math functions using two values entered sequentially into the display. If you are *keying in* both numbers, remember that they must be separated by [ENTER] or any other function—like [g] [INT] or [1/x]—that terminates digit entry.

For a two-number function, the first value entered is considered the y-value because it is placed into the Y-register for memory storage. The second value entered is considered the x-value because it remains in the display, which is the X-register.

The arithmetic operators, [+], [−], [×], and [÷], are the four basic two-number functions. Others are given below.

The Power Function

Pressing [y^x] calculates the value of y raised to the x power. The base number, y, is keyed in before the exponent, x.

To Calculate	Keystrokes	Display
$2^{1.4}$	2 [ENTER] 1.4 [y^x]	**2.6390**
$2^{-1.4}$	2 [ENTER] 1.4 [CHS] [y^x]	**0.3789**
$(-2)^3$	2 [CHS] [ENTER] 3 [y^x]	**−8.0000**
$\sqrt[3]{2}$ or $2^{1/3}$	2 [ENTER] 3 [1/x] [y^x]	**1.2599**

Percentages

The percentage functions, [%] and [Δ%], preserve the value of the original base number along with the result of the percentage calculation. As shown in the example below, this allows you to carry out subsequent calculations using the base number and the result without re-entering the base number.

Percent. The [%] function calculates the specified percentage of a base number.

For example, to find the sales tax at 3% and total cost of a $15.76 item:

Keystrokes	Display	
15.76 [ENTER]	**15.7600**	Enters the base number (the price).
3 [g] [%]	**0.4728**	Calculates 3% of $15.76 (the tax).
[+]	**16.2328**	Total cost of item ($15.76 + $0.47).

Percent Difference. The [Δ%] function calculates the percent *difference* between two numbers. The result expresses the relative increase (a positive result) or decrease (a negative result) of the second number entered compared to the first number entered.

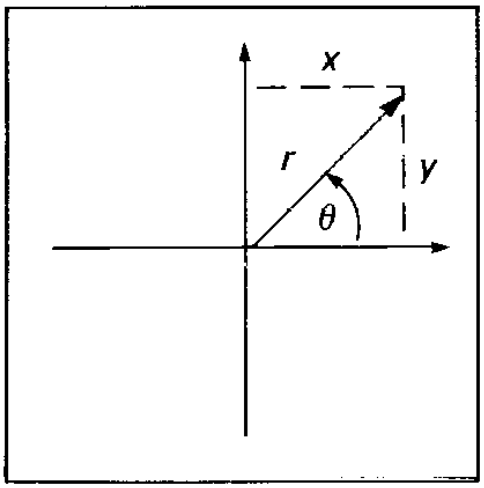
For example, suppose the $15.76 item only cost $14.12 last year. What is the percent difference in last year's price relative to *this* year's?

Keystrokes	Display	
15.76 [ENTER]	**15.7600**	This year's price (our base number).
14.12 [g] [Δ%]	**-10.4061**	*Last* year's price was 10.41% *less* than *this* year's price.

Polar and Rectangular Coordinate Conversions

The [→P] and [→R] functions are provided in the HP-15C for conversions between polar coordinates and rectangular coordinates.

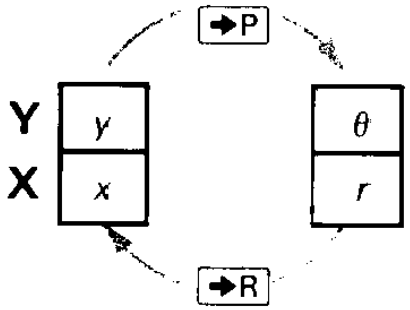
The angle θ is assumed to be in the units set by the current trigonometric mode, whether degrees (in a decimal format, not a minutes-seconds format), radians, or grads. θ is measured as shown in the illustration at right.



Polar Conversion. Pressing [g] [→P] (*polar*) converts a set of rectangular coordinates (x, y) to polar coordinates (magnitude r,

angle θ). The y-value must be entered first, the x-value second. Upon executing [g] [→P], r will appear in the display. Press [x≷y] (*X exchange Y*) to bring θ out of the Y-register and into the display (X-register). θ will be returned as a value between $-180°$ and $180°$, between $-\pi$ and π radians, or between -200 and 200 grads.

Rectangular Conversion. Pressing [f] [→R] (*rectangular*) converts a set of polar coordinates (magnitude r, angle θ) into rectangular coordinates (x, y). θ must be entered first, then r. Upon executing [f] [→R], x will be displayed first; press [x≷y] to display y.



Keystrokes	Display	
[g] [DEG]		Set to Degrees mode (no annunciator).
5 [ENTER]	5.0000	y-value.
10	10	x-value.
[g] [→P]	11.1803	r.
[x≷y]	26.5651	θ; rectangular coordinates converted *to* polar coordinates.
30 [ENTER]	30.0000	θ.
12	12	r.
[f] [→R]	10.3923	x-value.
[x≷y]	6.0000	y-value. Polar coordinates converted *to* rectangular coordinates.

Section 3

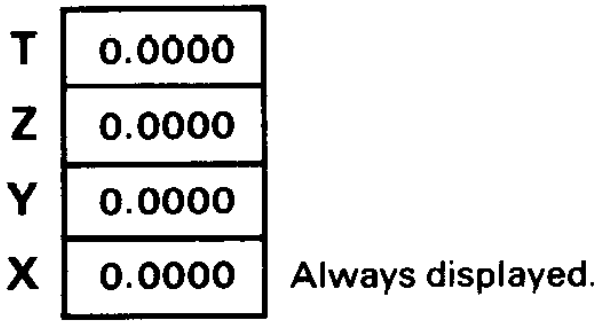
The Automatic Memory Stack, LAST X, and Data Storage

The Automatic Memory Stack and Stack Manipulation

HP operating logic is based on a mathematical logic known as "Polish Notation," developed by the noted Polish logician Jan Łukasiewicz (*Wookashye'veech*) (1878-1956). Conventional algebraic notation places the algebraic operators *between* the relevant numbers or variables when evaluating algebraic expressions. Łukasiewicz's notation specifies the operators *before* the variables. For optimal efficiency of calculator use, HP applied the convention of specifying (entering) the operators *after* specifying (entering) the variable(s). Hence the term "Reverse Polish Notation" (RPN).

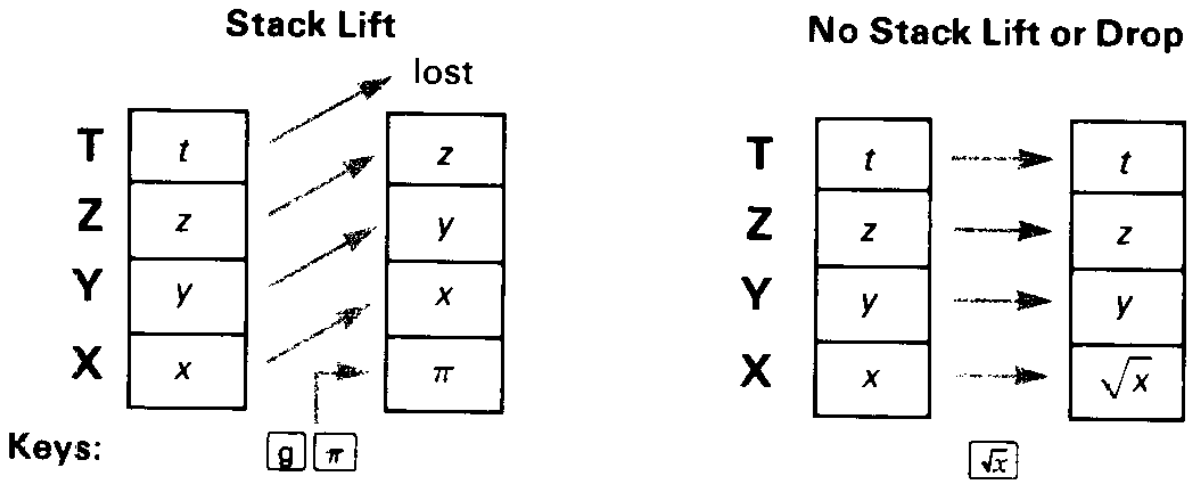
The HP-15C uses RPN to solve complicated calculations in a straightforward manner, without parentheses or punctuation. It does so by automatically retaining and returning intermediate results. This system is implemented through the automatic memory stack and the [ENTER] key, minimizing total keystrokes.

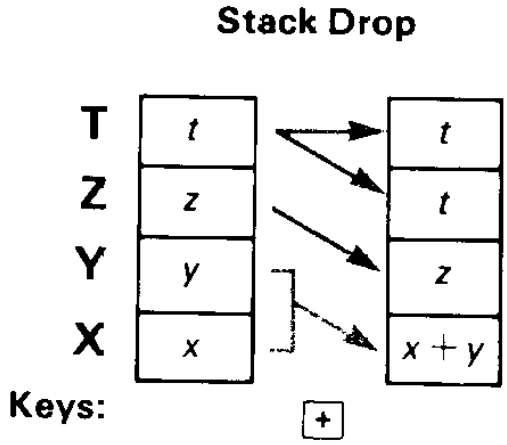
The Automatic Memory Stack Registers



When the HP-15C is in Run mode (no **PRGM** annunciator displayed), the number that appears in the display is the number in the X-register.

Any number that is keyed in or results from the execution of a numeric function is placed into the display (X-register). This action will cause numbers already in the stack to lift, remain in the same register, or drop, depending upon both the immediately preceding and the current operation. Numbers in the stack are stored on a last-in, first-out basis. The three stacks drawn below illustrate the three types of stack movement. Assume x, y, z, and t represent any numbers which may be in the stack.



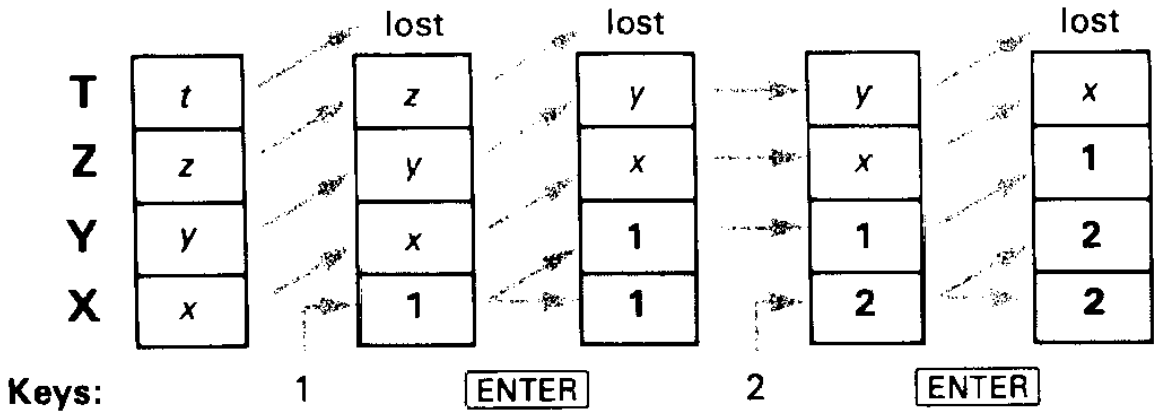


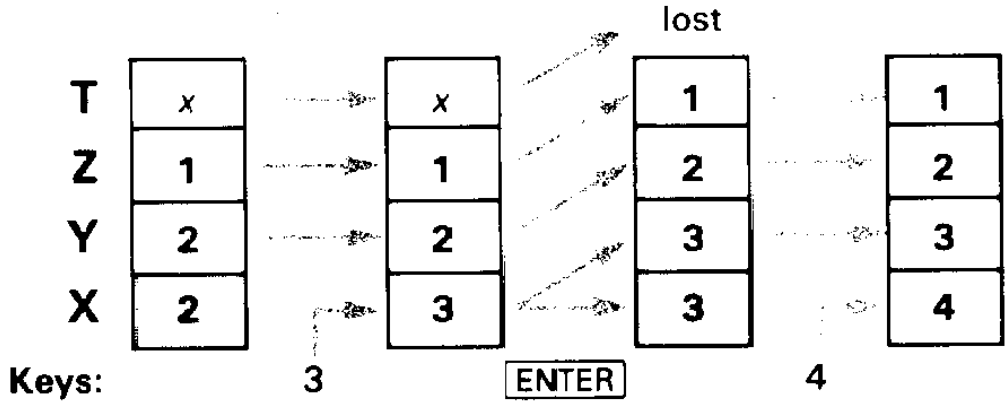
Notice the number in the T-register remains there when the stack drops, allowing this number to be used repetitively as an arithmetic constant.

Stack Manipulation Functions

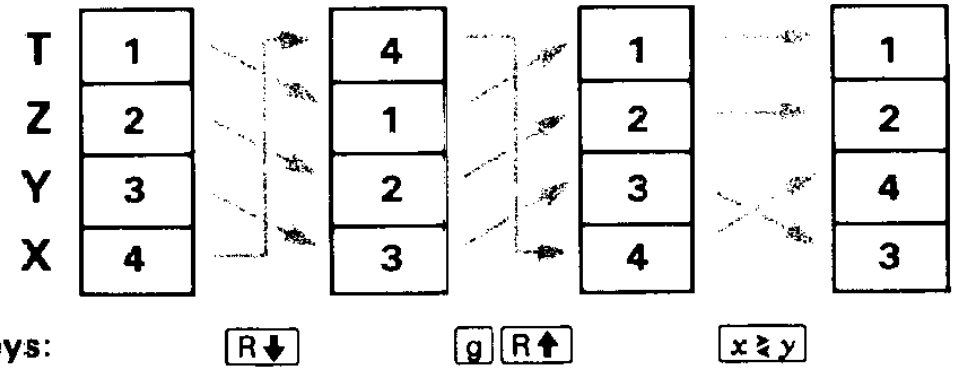
ENTER. Pressing ENTER separates two numbers keyed in one after the other. It does so by lifting the stack and copying the number in the display (X-register) into the Y-register. The next number entered then writes over the value in the X-register; there is no stack lift. The example below shows what happens as the stack is

filled with the numbers 1, 2, 3, 4. (The shading indicates that the contents of that register will be written over when the next number is keyed in or recalled.)



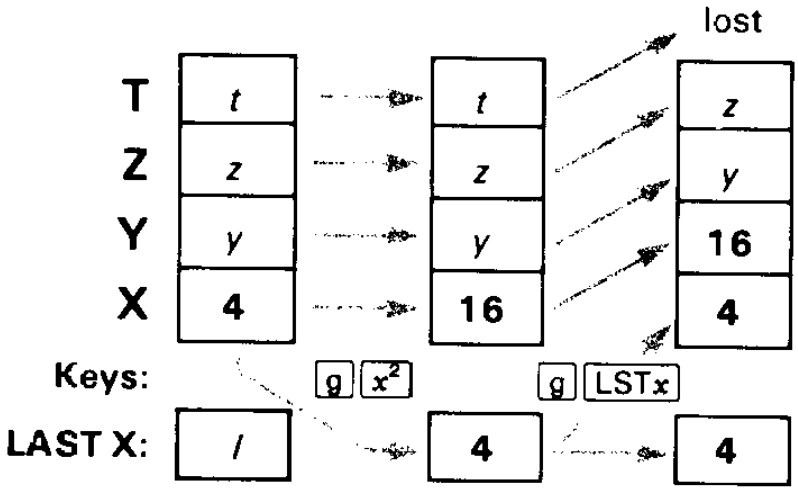


R↓ (***roll down***), R↑ (***roll up***), **and** x≷y (***X exchange Y***). R↓ and R↑ roll the contents of the stack registers up or down one register (one value moves between the X- and the T-register). No values are lost. x≷y exchanges the numbers in the X- and Y-registers. If the stack were loaded with the sequence 1, 2, 3, 4, the following shifts would result from pressing R↓, R↑, and x≷y.



The LAST X Register and [LSTx]

The LAST X register, a separate memory register, preserves the value that was last in the display *before execution of a numeric operation*.* Pressing [g] [LSTx] (*LAST X*) places a copy of the contents of the LAST X register into the display (X-register). For example:



The [LSTx] feature saves you from having to re-enter numbers you want to use again (as shown under Arithmetic Calculations With Constants, page 39). It can also assist you in error recovery, such as executing the wrong function or keying in the wrong number.

For example, suppose you mistakenly entered the wrong divisor in a chain calculation:

Keystrokes	Display	
287 [ENTER]	287.0000	
12.9 [÷]	22.2481	Oops! The wrong divisor.
[g] [LSTx]	12.9000	Retrieves from LAST X the last entry to the X-register (the incorrect divisor) before [÷] was executed.

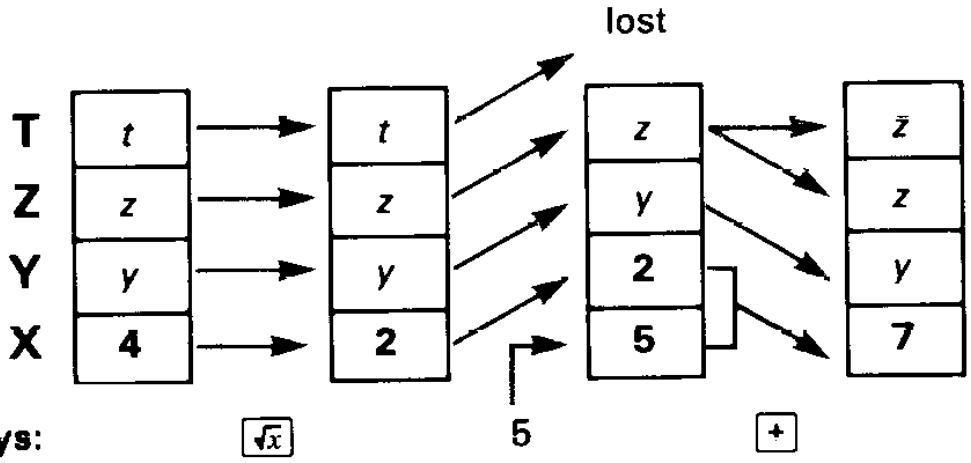
* Unless that operation was [x̄], [s], or [L.R.], which don't use or preserve the value in the display (X-register), but instead calculate from data in the statistics storage registers (R_2 to R_7). For a complete list of operations which save x in LAST X, refer to appendix B.

Keystrokes	Display	
[×]	**287.0000**	Reverses the function that produced the wrong answer.
13.9 [÷]	**20.6475**	The correct answer.

Calculator Functions and the Stack

When you want to key in two numbers, one after the other, you press [ENTER] between entries of the numbers. However, when you want to key in a number immediately following any function (including manipulations like [R↓]), you do not need to use [ENTER]. Why? Executing most HP-15C functions has this additional effect:

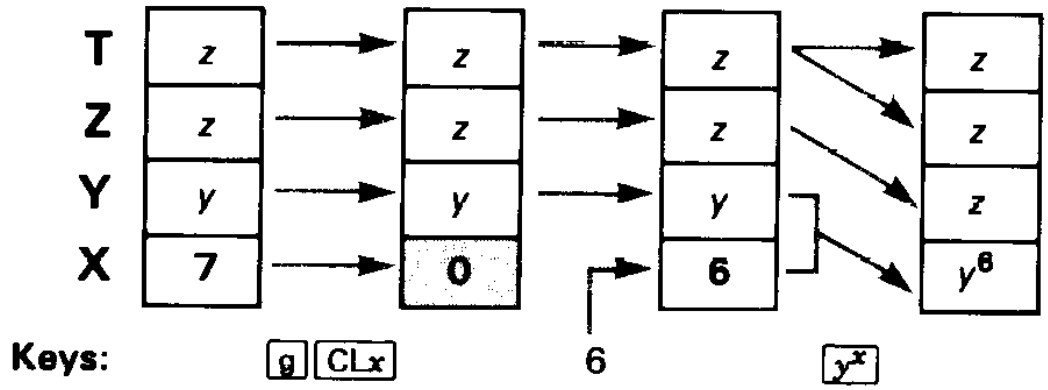
- The automatic memory stack is lift-*enabled*; that is, the stack will lift automatically when the *next* number is keyed or recalled into the display.
- Digit entry is terminated, so the next number starts a new entry.



There are four functions—[ENTER], [CLx], [Σ+], and [Σ-]—that *disable* stack lift.* They do *not* provide for the lifting of the stack when the *next* number is keyed in or recalled. Following the execution of one of these functions, a new number will simply write over the currently displayed number instead of causing the stack to lift. (Although the stack lifts when [ENTER] is pressed, it will *not* lift when the *next* number is keyed in or recalled. The operation of

* [←] will also disable the stack lift *if digit entry is terminated*, making [←] clear the entire display like [CLx]. Otherwise, it is neutral. For a further discussion of the stack, refer to appendix B.

[ENTER] illustrated on page 34 shows how [ENTER] thus disables the stack.) In most cases, the above effects will come so naturally that you won't even think about them.

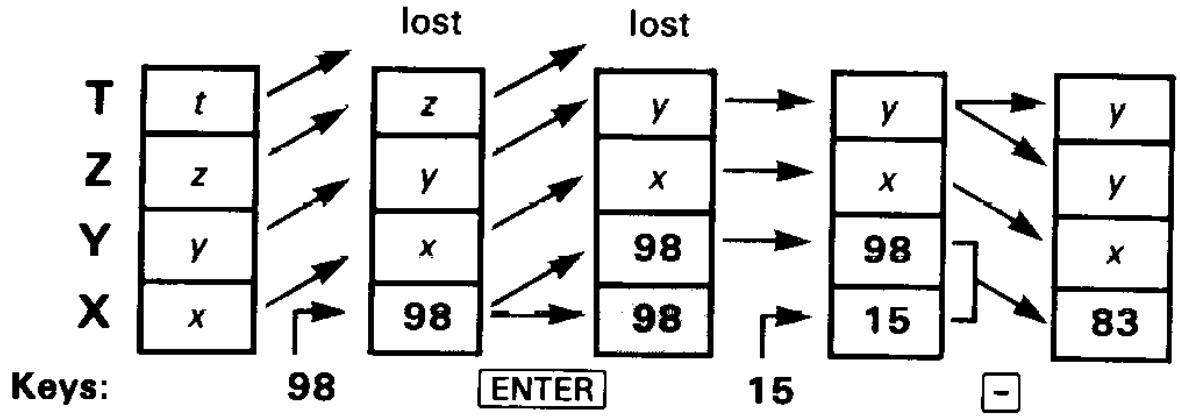


Order of Entry and the [ENTER] Key

An important aspect of two-number functions is the positioning of the numbers in the stack. To execute an arithmetic function, the numbers should be positioned in the stack in the same way that you would vertically position them on paper. For example:

98	98	98	98
−15	+15	×15	15

As you can see, the first (or top) number would be in the Y-register, while the second (or bottom) number would be in the X-register. When the mathematics operation is performed, the stack drops, leaving the result in the X-register. Here is how a subtraction operation is executed in the calculator:



The same number positioning would be used to add 15 to 98, multiply 98 by 15, or divide 98 by 15.

Nested Calculations

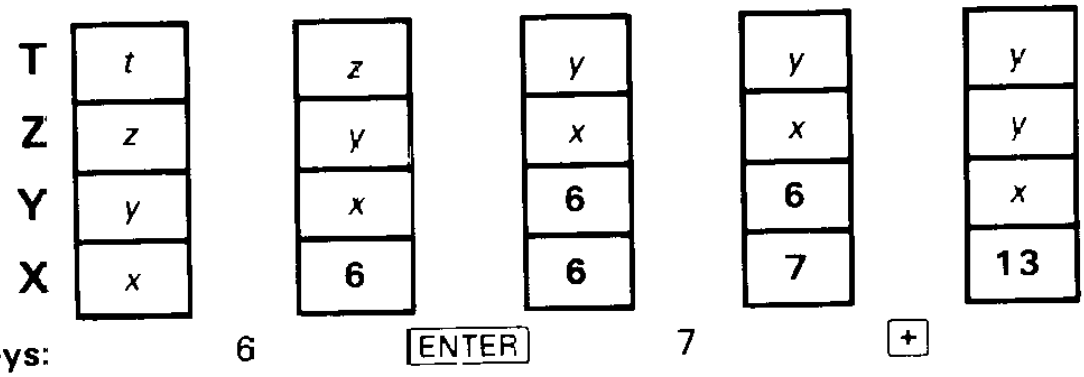
The automatic stack lift and stack drop make it possible to do nested calculations without using parentheses or storing intermediate results. A nested calculation is solved simply as a series of one- and two-number operations.

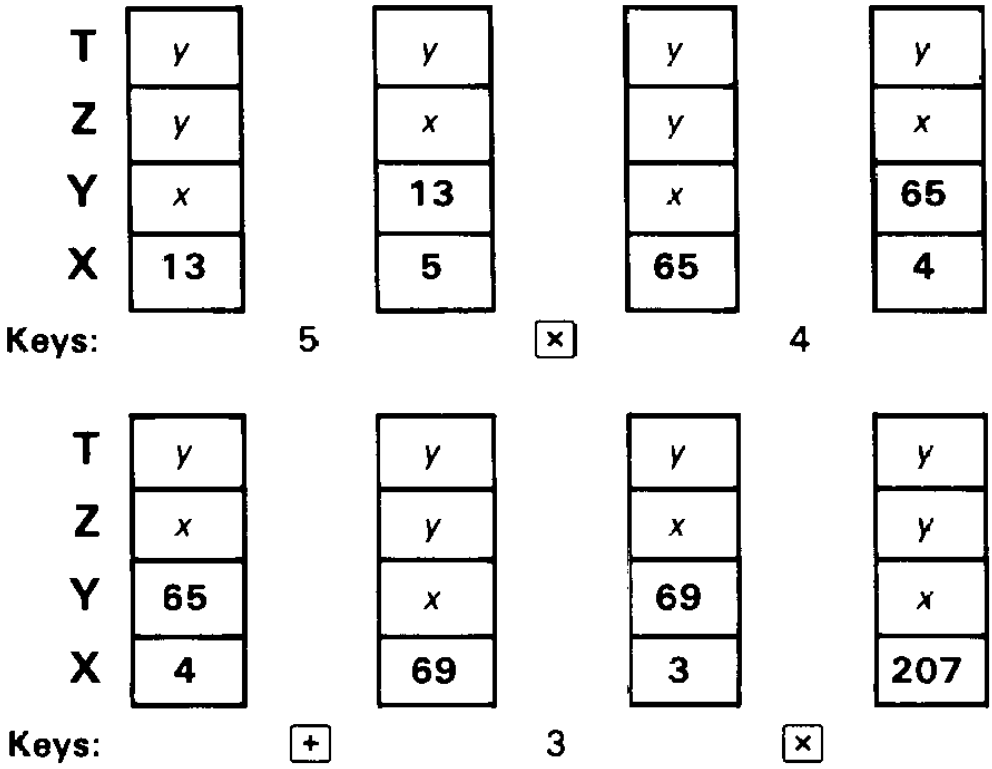
Almost every nested calculation you are likely to encounter can be done using just the four stack registers. It is usually wisest to begin the calculation at the innermost number or pair of parentheses and work outward (as you would for a manual calculation). Otherwise, you may need to place an intermediate result into a storage register. For example, consider the calculation of

$$3\,[4 + 5\,(6 + 7)]\ :$$

Keystrokes	Display	
6 [ENTER] 7 [+]	**13.0000**	Intermediate result of $(6 + 7)$.
5 [×]	**65.0000**	Intermediate result of $5\,(6 + 7)$.
4 [+]	**69.0000**	Intermediate result of $[4 + 5\,(6 + 7)]$.
3 [×]	**207.0000**	Final result: $3\,[4 + 5\,(6 + 7)]$.

The following sequence illustrates the stack manipulation in this example. The stack automatically drops after each two-number calculation, and then lifts when a new number is keyed in. (For simplicity, throughout the rest of this handbook we will not show arrows between the stacks.)





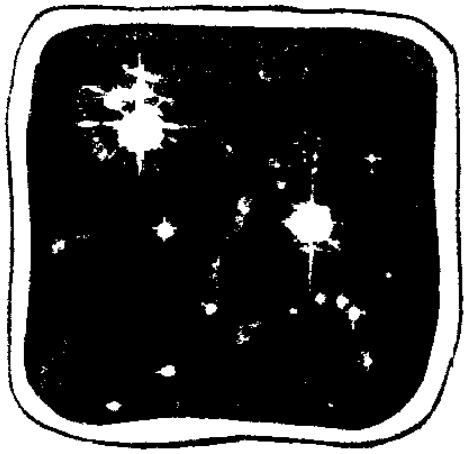
Arithmetic Calculations With Constants

There are three ways (without using a storage register) to manipulate the memory stack to perform repeated calculations with a constant:

1. Use the LAST X register.
2. Load the stack with a constant and operate upon different numbers. (Clear the X-register every time you want to change the number operated upon.)
3. Load the stack with a constant and operate upon an *accumulating* number. (Do *not* change the number in the X-register.)

LAST X. Use your constant in the X-register (that is, enter it second) so that it always will be saved in the LAST X register. Pressing [g] [LSTx] will retrieve the constant and place it into the X-register (the display). This can be done repeatedly.

Example: Two close stellar neighbors of Earth are Rigel Centaurus (4.3 light-years away) and Sirius (8.7 light-years away). Use the speed of light, c (3.0×10^8 meters/second, or 9.5×10^{15} meters/year), to figure the distances to these stars in meters. (The stack diagrams show only one decimal place.)

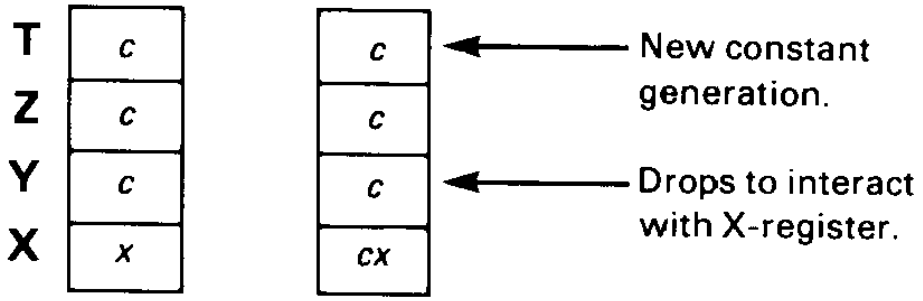


T	t	z	y	y
Z	z	y	x	x
Y	y	x	4.3	4.3
X	x	4.3	4.3	9.5 15
Keys:		4.3	ENTER	9.5 EEX 15
LAST X:	/	/	/	/

T	y	y	y	x
Z	x	y	x	4.1 16
Y	4.3	x	4.1 16	8.7
X	9.5 15	4.1 16	8.7	9.5 15
Keys:		×	8.7	g LSTx
LAST X:	/	9.5 15	9.5 15	9.5 15

T	x	x	
Z	4.1 16	x	
Y	8.7	4.1 16	← (Rigel Centaurus is 4.1×10^{16} meters away.)
X	9.5 15	8.3 16	← (Sirius is 8.3×10^{16} meters away.)
Keys:		×	
LAST X:	9.5 15	9.5 15	

Loading the Stack with a Constant. Because the number in the T-register is replicated when the stack drops, this number can be used as a constant in arithmetic operations.



Keys: [×]

Fill the stack with a constant by keying it into the display and pressing [ENTER] three times. Key in your initial argument and perform the arithmetic operation. The stack will drop, a copy of the constant will "fall" into the Y-register, and a new copy of the constant will be generated in the T-register.

If the variables change (as in the preceding example), be sure and clear the display before entering the new variable. This disables the stack so that the arithmetic result will be written over and only the constant will occupy the rest of the stack.

If you do *not* have different arguments, that is, the operation will be performed upon a *cumulative* number, then do *not* clear the display—simply repeat the arithmetic operation.

Example: A bacteriologist tests a certain strain of microorganisms whose population typically increases by 15% each day (a growth factor of 1.15). If she starts with a sample culture of 1000, what will be the bacteria population at the end of each day for four consecutive days?

Keystrokes	Display	
1.15	**1.15**	Growth factor.
[ENTER] [ENTER] [ENTER]	**1.1500**	Filling the stack.
1000	**1,000**	Initial culture size.

Keystrokes	Display	
[×]	1,150.0000	Population at the end of day 1.
[×]	1,322.5000	Day 2.
[×]	1,520.8750	Day 3.
[×]	1,749.0063	Day 4.

Storage Register Operations

When numbers are stored or recalled, they are copied between the display (X-register) and the data storage registers. At "power-up" (initial turn-on or Continuous Memory reset) the HP-15C has 21 directly accessible storage registers: R_0 through R_9, $R_{.0}$ through $R_{.9}$, and the Index register (R_I) (see the diagram of the registers on the inside back cover). Six registers, R_2 to R_7, are also used for statistics calculations.

The number of available data storage registers can be increased or decreased. The [DIM] function, which is used to reallocate registers in calculator memory, is discussed in appendix C, Memory Allocation. The lowest-numbered registers are the last to be deallocated from data storage, *therefore it is wisest to store data in the lowest-numbered registers available.*

Storing and Recalling Numbers

[STO] (*store*). When followed by a storage register address (0 through 9 or .0 through .9*), this function copies a number from the display (X-register) into the specified data storage register. It will replace any existing contents of that register.

[RCL] (*recall*). Similarly, you can recall data from a particular register into the display by pressing [RCL] followed by the register address. This brings a *copy* of the desired data into the display; the contents of the storage register remain unaltered.

[x≷] (*X exchange*). Followed by 0 through .9,* this function *exchanges the contents* of the X-register and the addressed data storage register. This is useful to view storage registers without disturbing the stack.

---

* All storage register operations can also be performed with the Index register (using [I] or [(i)]), which is covered in section 10, and with matrices, section 12.

The above are stack lift-enabling operations, so the number remaining in the X-register can be used for subsequent calculations. If you address a nonexistent register, the display will show **Error 3**.

Example: Springtime is coming and you want to keep track of 24 crocuses planted in your garden. Store the number of crocuses blooming the first day, and add to this the number of new blooms the second day.

Keystrokes	Display	
3 [STO] 0	**3.0000**	Stores the number of first-day blooms in R_0.

Turn the calculator off. Next day, turn it back on again.

[RCL] 0	**3.0000**	Recalls the number of crocuses that bloomed yesterday.
5 [+]	**8.0000**	Adds today's new blooms to get the total blooming crocuses.

Clearing Data Storage Registers

Pressing [f] CLEAR [REG] (*clear registers*) clears the contents of *all* data storage registers to zero. (It does not affect the stack or the LAST X register.) To clear a single data storage register, store zero in that register. Resetting Continuous Memory clears all registers and the stack.

Storage and Recall Arithmetic

Storage Arithmetic. Suppose you not only wanted to store a number, but perform arithmetic with it and store the result in the same register. You can do this directly—without using [RCL]—by using the following procedure:

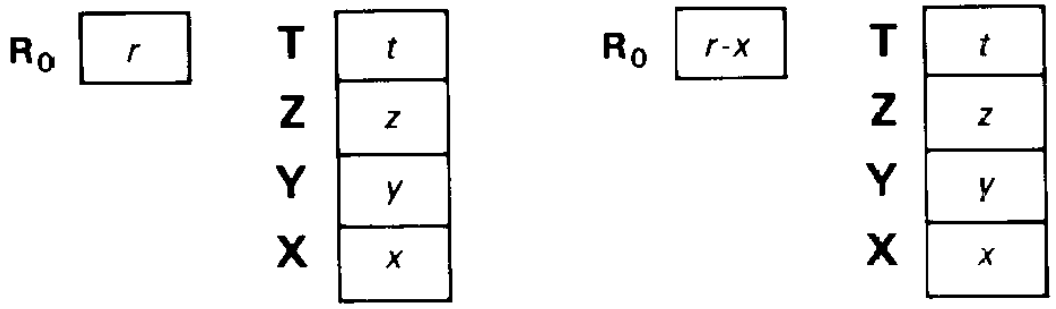
1. Have your second operand (besides the one in storage) in the display (as the result of a calculation, a recall, or keying in).
2. Press [STO].

3. Press [+], [−], [×], or [÷].
4. Key in the register address (0 to 9, .0 to .9). (The Index register, discussed in section 10, can also be used.)

The new number in the register is determined as follows:

For storage arithmetic,

$$\text{new contents of register} = \text{old contents of register} \begin{matrix} + \\ - \\ \times \\ \div \end{matrix} \text{number in display}$$

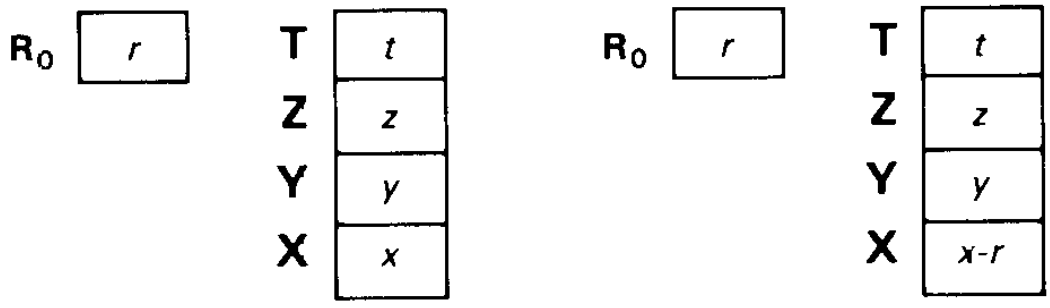


Keys: [STO] [−] 0

Recall Arithmetic. Recall arithmetic allows you to perform arithmetic with the displayed value and a stored value *without lifting the stack*, that is, without losing any values from the Y-, Z-, and T-registers. The keystroke sequence is the same as for storage arithmetic using [RCL] in place of [STO].

For recall arithmetic,

$$\text{new display} = \text{old display} \begin{matrix} + \\ - \\ \times \\ \div \end{matrix} \text{contents of register}$$



Keys: [RCL] [−] 0

Example: Keep a running count of your newly blooming crocuses for two more days.

Keystrokes	Display	
8 [STO] 0	**8.0000**	Places the total number of blooms as of day 2 in R_0.
4 [STO] [+] 0	**4.0000**	Day 3: adds four new blooms to those already blooming.
3 [STO] [+] 0	**3.0000**	Day 4: adds three new blooms.
24 [RCL] [−] 0	**9.0000**	Subtracts total number of blooms summed in R_0 (15) from the total number of plants (24): 9 crocuses have not bloomed.
[RCL] 0	**15.0000**	(The number in R_0 does not change.)

Overflow and Underflow

If an attempted storage or recall arithmetic operation would result in overflow in a data storage register, the value in the affected register will be replaced with $\pm 9.999999999 \times 10^{99}$ and the display will blink. To stop the blinking (clear the overflow condition), press [←] or [ON] or [g] [CF] 9.

In case of underflow, the value in the register will be replaced with zero (no display blinking). Overflow and underflow are discussed further on page 61.

Problems

1. Calculate the value of x in the following equation.

$$x = \sqrt{\frac{8.33\,(4 - 5.2) \div [(8.33 - 7.46)\,0.32]}{4.3\,(3.15 - 2.75) - (1.71)\,(2.01)}}$$

Answer: 4.5728.

A possible keystroke solution is:

4 [ENTER] 5.2 [−] 8.33 [×] [g] [LSTx] 7.46 [−] 0.32 [×] [÷] 3.15 [ENTER] 2.75 [−] 4.3 [×] 1.71 [ENTER] 2.01 [×] [−] [÷] [√x]

2. Use arithmetic with constants to calculate the remaining balance of a $1000 loan after six payments of $100 each and an interest rate of 1% (0.01) per payment period.

   Procedure: Load the stack with $(1 + i)$, where i = interest rate, and key in the initial loan balance. Use the following formula to find the new balance after each payment.

   $$\text{New Balance} = ((\text{Old Balance}) \times (1 + i)) - \text{Payment}$$

   The first part of the key sequence would be:

   1.01 [ENTER] [ENTER] [ENTER] 1000

   For each payment, execute:

   [×] 100 [−]

   Balance after six payments: $446.32.

3. Store 100 in R_5. Then:

   1. Divide the contents of R_5 by 25.
   2. Subtract 2 from the contents of R_5.
   3. Multiply the contents of R_5 by 0.75.
   4. Add 1.75 to the contents of R_5.
   5. Recall the contents of R_5.

   Answer: 3.2500.

Section 4

Statistics Functions

A word about the statistics functions: their use is based on an understanding of memory stack operation (section 3). You will find that order of entry is important for most statistics calculations.

Probability Calculations

The input for permutation and combination calculations is restricted to *nonnegative integers*. Enter the y-value before the x-value. These functions, like the arithmetic operators, cause the stack to drop as the result is placed in the X-register.

Permutations. Pressing [f] [Py,x] calculates the number of possible different *arrangements* of y different items taken in quantities of x items at a time. No item occurs more than once in an arrangement, and different orders of the same x items in an arrangement *are* counted separately. The formula is

$$P_{y,x} = \frac{y!}{(y-x)!}$$

Combinations. Pressing [g] [Cy,x] calculates the number of possible *sets* of y different items taken in quantities of x items at a time. No item occurs more than once in a set, and different orders of the same x items in a set are *not* counted separately. The formula is

$$C_{y,x} = \frac{y!}{x!\,(y-x)!}$$

Examples: How many different arrangements are possible of five pictures which can be hung on the wall three at a time?

Keystrokes	Display	
5 [ENTER] 3	**3**	Five (y) pictures put up three (x) at a time.
[f] [Py,x]	**60.0000**	Sixty different arrangements possible.

How many different four-card hands can be dealt from a deck of 52 cards?

Keystrokes	Display	
52 [ENTER] 4	**4**	Fifty-two (y) cards dealt four (x) at a time.
[g] [Cy,x]	**270,725.0000**	Number of different hands possible.

The execution times for these functions may last several seconds, depending on the magnitude of the x and y inputs. The display will show **running** during this time. The maximum size of x or y is 9,999,999,999.

Random Number Generator

Pressing [f] [RAN#] (*random number*) will generate a random number (part of a uniformly distributed pseudo-random number sequence) in the range $0 \leqslant r < 1$.*

At initial power-up (including reset of Continuous Memory), the HP-15C random number generator will use zero as a "seed" to initiate a random number sequence. Any time you generate a random number, that number becomes the seed for the next random number. You can initiate a different random number sequence by storing a new seed for the random number generator. (Repetition of a random number seed will produce repetition of the random number sequence.)

[STO] [f] [RAN#] will store the X-register number ($0 \leqslant r < 1$) as a new seed for the random number generator. (A value for r outside this range will be converted to fit within the range.)

[RCL] [f] [RAN#] will recall to the display the current random number seed.

---

* Passes the spectral test (D. Knuth, *Seminumerical Algorithms*, vol. 2, 1969).

Keystrokes	Display	
.5764	**0.5764**	Stores 0.5764 as random number seed. (The [f] keystroke may be omitted.)
[STO] [f] [RAN#]	**0.5764**	
[f] [RAN#]	**0.3422**	Random number sequence initiated by the above seed.
[f] [RAN#]	**0.2809**	
[←]	**0.0000**	
[RCL] [f] [RAN#]	**0.2809**	Recalls last random number generated, which is the new seed. (The [f] may be omitted.)

Accumulating Statistics

The HP-15C performs one- and two-variable statistical calculations. The data is first entered into the Y- and X-registers. Then the [Σ+] function automatically calculates and stores statistics of the data in storage registers R_2 through R_7. These registers are therefore referred to as the *statistics registers*.

Before beginning to accumulate statistics for a new set of data, press [f] CLEAR [Σ] to clear the statistics registers and stack. (If you have reallocated registers in memory and any of the statistics registers no longer exist, **Error 3** will be displayed when you try to use CLEAR [Σ], [Σ+], or [Σ-]. Appendix C explains how to reallocate memory.)

In one-variable statistical calculations, enter each data point (x-value) by keying in x and then press [Σ+].

In two-variable statistical calculations, enter each data pair (the x- and y-values) as follows:

1. Key y into the display first.
2. Press [ENTER]. The displayed y-value is copied into the Y-register.
3. Key x into the display.
4. Press [Σ+]. The current number of accumulated data points, n, will be displayed. The x-value is saved in the LAST X register and y remains in the Y-register. [Σ+] disables stack lift, so the stack will not lift when the next number is keyed in.

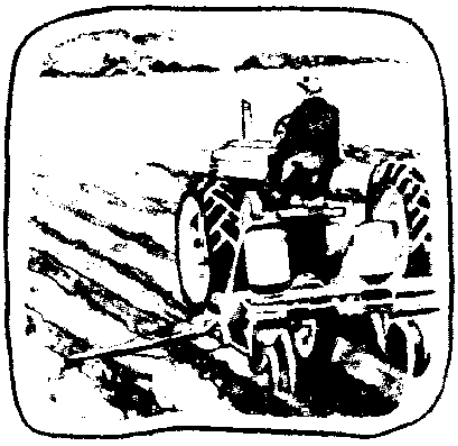
In some cases involving x or y data values that differ by a relatively small amount, the calculator cannot compute s, r, linear regression, or $\hat{y}$, and will display **Error 2**. This will not happen, however, if you normalize the data by keying in only the difference between each value and the mean or approximate mean of the values. This difference must be added back to the calculations of $\bar{x}$, $\hat{y}$, and the y-intercept ([L.R.]). For example, if your x-values were 665999, 666000, and 666001, you should enter the data as -1, 0, and 1; then add 666000 back to the relevant results.

The statistics of the data are compiled as follows:

Register	Contents	
R_2	n	Number of data points accumulated (n also appears in the X-register).
R_3	Σx	Summation of x-values.
R_4	Σx^2	Summation of squares of x-values.
R_5	Σy	Summation of y-values.
R_6	Σy^2	Summation of squares of y-values.
R_7	Σxy	Summation of products of x- and y-values.

You can recall any of the accumulated statistics to the display (X-register) by pressing [RCL] and the number of the data storage register containing the desired statistic. If you press [RCL] [Σ+], Σx and Σy will be copied simultaneously from R_3 and R_5, respectively, into the X-register and the Y-register, respectively. (The sequence [RCL] [Σ+] lifts the stack twice if stack lift is enabled, once if not, and then enables stack lift.)

Example: Agronomist Silas Farmer has developed a new variety of high-yield rice, and has measured the plant's yield rate as a function of fertilization. Use the [Σ+] function to accumulate the data below to find the values for Σx, Σx^2, Σy, Σy^2, and Σxy for nitrogen fertilizer application (x) versus grain yield (y).



X	NITROGEN APPLIED (kg per hectare*), x	0.00	20.00	40.00	60.00	80.00
Y	GRAIN YIELD (metric tons per hectare), y	4.63	4.78	6.61	7.21	7.78

* A hectare equals 2.47 acres.

Keystrokes	Display	
f CLEAR Σ	**0.0000**	Clears statistical storage registers (R_2 through R_7 and the stack).
f FIX 2	**0.00**	Limits display to two decimal places, like the data.
4.63 ENTER	**4.63**	
0 Σ+	**1.00**	First data point.
4.78 ENTER	**4.78**	
20 Σ+	**2.00**	Second data point.
6.61 ENTER	**6.61**	
40 Σ+	**3.00**	Third data point.
7.21 ENTER	**7.21**	
60 Σ+	**4.00**	Fourth data point.
7.78 ENTER	**7.78**	
80 Σ+	**5.00**	Fifth data point.
RCL 3	**200.00**	Sum of x-values, Σx (kg of nitrogen).
RCL 4	**12,000.00**	Sum of squares of x-values, Σx^2.
RCL 5	**31.01**	Sum of y-values, Σy (grain yield).
RCL 6	**200.49**	Sum of squares of y-values, Σy^2.
RCL 7	**1,415.00**	Sum of products of x- and y-values, Σxy.

Correcting Accumulated Statistics

If you discover that you have entered data incorrectly, the accumulated statistics can be easily corrected. Even if only one value of an (x,y) data pair is incorrect, you must delete and re-enter *both* values.

1. Key the *incorrect* data pair into the Y- and X-registers.
2. Press [g] [Σ-] to delete the incorrect data.
3. Key in the correct values for x and y.
4. Press [Σ+].

Alternatively, if the incorrect data point or pair is the most recent one entered and [Σ+] has been pressed, you can press [g] [LSTx] [g] [Σ-] to remove the incorrect data.*

Example: After keying in the preceding data, Farmer realizes he misread a smeared figure in his lab book. The second y-value should have been 5.78 instead of 4.78. Correct the data input.

Keystrokes	Display	
4.78 [ENTER]	**4.78**	Keys in the data pair we
20 [g] [Σ-]	**4.00**	want to replace and deletes the accompanying statistics. The n-value drops to four.
5.78 [ENTER]	**5.78**	Keys in and accumulates the replacement data pair.
20 [Σ+]	**5.00**	The n-value is back to five.

We will use these statistics in the rest of the examples in this section.

---

* Note that these methods of data deletion will not delete any rounding errors that may have been generated in the statistics registers. This difference will not be serious unless the erroneous pair has a magnitude that is enormous compared with the correct pair; in such a case, it would be wise to start over!

Mean

The [x̄] function computes the arithmetic mean (average) of the x-and y-values using the formulas shown in appendix A and the statistics accumulated in the relevant registers. When you press [g] [x̄], the contents of the stack lift (two registers if stack lift is enabled, one if not); the mean of x ($\bar{x}$) is copied into the X-register as the mean of y ($\bar{y}$) is copied simultaneously into the Y-register. Press [x≷y] to view $\bar{y}$.

Example: From the corrected statistics data we have already entered and accumulated, calculate the average fertilizer application, $\bar{x}$, and average grain yield $\bar{y}$, for the entire range.

Keystrokes	Display	
[g] [x̄]	40.00	Average kg of nitrogen, $\bar{x}$, for all cases.
[x≷y]	6.40	Average tons of rice, $\bar{y}$, for all cases.

Standard Deviation

Pressing [g] [s] computes the *standard deviation* of the accumulated statistics data. The formulas used to compute s_x, the standard deviation of the accumulated x-values, and s_y, the standard deviation of the accumulated y-values, are given in appendix A.

This function gives an estimate of the population standard deviation from the sample data, and is therefore termed the *sample* standard deviation.* When you press [g] [s], the contents of the stack registers are lifted (twice if stack lift is enabled, once if not); s_x is placed into the X-register and s_y is placed into the Y-register. Press [x≷y] to view s_y.

---

* When your data constitutes not just a sample of a population but *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 $\sqrt{(n-1)/n}$ from the formula used for the [s] function. The difference between the values is small for large n, and for most applications can be ignored. But if you want to calculate the exact value of the population standard deviation for an entire population, you can easily do so: simply add, using [Σ+], the mean ($\bar{x}$) of the data to the data before pressing [g] [s]. The result will be the population standard deviation. (If you subsequently correct any of your accumulated data values, remember to delete the first mean value and add the corrected one.)

Example: Calculate the standard deviation about the mean calculated above.

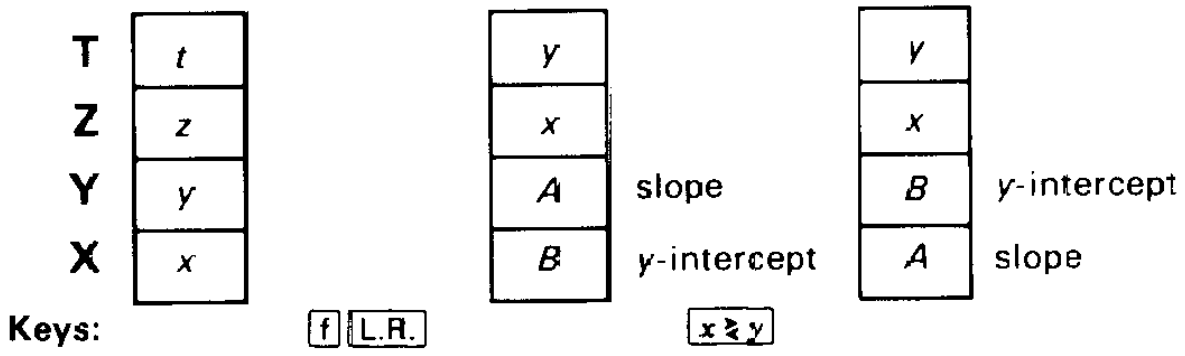
Keystrokes	Display	
g s	**31.62**	Standard deviation about the mean nitrogen application, $\bar{x}$.
x≷y	**1.24**	Standard deviation about the mean grain yield, $\bar{y}$.

Linear Regression

Linear regression is a statistical method for finding a straight line that best fits a set of two or more data pairs, thus providing a relationship between two variables. By the method of least squares, f L.R. will calculate the slope, A, and y-intercept, B, of the linear equation:

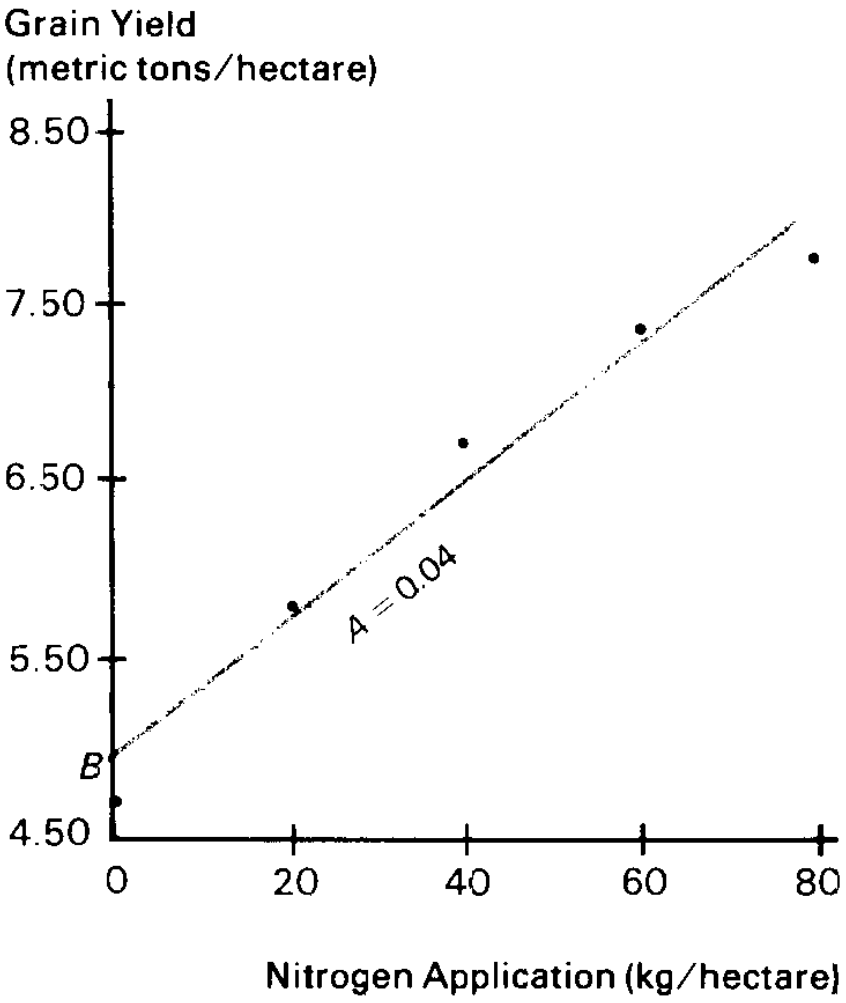
$$y = Ax + B$$

1. Accumulate the statistics of your data using the Σ+ key.
2. Press f L.R. The y-intercept, B, appears in the display (X-register). The slope, A, is copied simultaneously into the Y-register.
3. Press x≷y to view A. (As is the case with the functions x̄ and s, L.R. causes the stack to lift two registers if it's enabled, one if not.)



The slope and y-intercept of the least squares line of the accumulated data are calculated using the equations shown in appendix A.

Example: Find the y-intercept and slope of the linear approximation of the data and compare to the plotted data on the graph below.



Keystrokes	Display	
f L.R.	**4.86**	y-intercept of the line.
x≷y	**0.04**	Slope of the line.

Linear Estimation and Correlation Coefficient

When you press f ŷ,r the *linear estimate*, $\hat{y}$, is placed in the X-register and the *correlation coefficient*, r, is placed in the Y-register. To display r, press x≷y.

Linear Estimation. With the statistics accumulated, an estimated value for y, denoted $\hat{y}$, can be calculated by keying in a proposed value for x and pressing [f][ŷ,r].

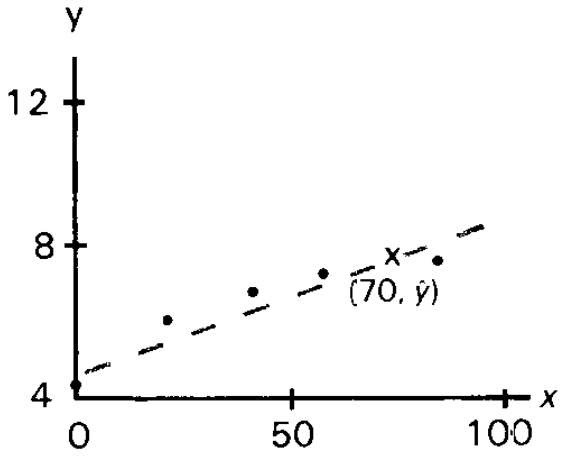
An Estimated value for x (denoted $\hat{x}$) can be calculated as follows:

1. Press [f][L.R.].
2. Key in the known y-value.
3. Press [x≷y][−][x≷y][÷].

Correlation Coefficient. Both linear regression and linear estimation presume that the relationship between the x and y data values can be approximated by a linear function. The correlation coefficient, r, is a determination of how closely your data fit a straight line. The range is $-1 \leqslant r \leqslant 1$, with -1 representing a perfectly negative correlation and $+1$ representing a perfectly positive correlation.

Note that if you do not key in a value for x before executing [f][ŷ,r], the number previously in the X-register will be used (usually yielding a meaningless value for $\hat{y}$).

Example: What if 70 kg of nitrogen fertilizer were applied to the rice field? Predict the grain yield based on Farmer's accumulated statistics. Because the correlation coefficient is automatically included in the calculation, you can view how closely the data fit a straight line by pressing [x≷y] after the y prediction appears in the display.



Keystrokes	Display	
70 f ŷ,r	**7.56**	Predicted grain yield in tons/hectare.
x≷y	**0.99**	The original data closely approximates a straight line.

Other Applications

Interpolation. Linear interpolation of tabular values, such as in thermodynamics and statistics tables, can be carried out very simply on the HP-15C by using the ŷ,r function. This is because linear interpolation *is* linear estimation: two consecutive tabular values are assumed to form two points on a line, and the unknown intermediate value is assumed to fall on that same line.

Vector Arithmetic. The statistical accumulation functions can be used to perform vector addition and subtraction. Polar vector coordinates must be converted to rectangular coordinates upon entry (θ, ENTER, r →R, Σ+). The results are recalled from R_3 (Σx) and R_5 (Σy) (using RCL Σ+) and converted back to polar coordinates, if necessary. Remember that for polar coordinates the angle is between $-180°$ and $180°$ (or $-\pi$ and π radians, or -200 and 200 grads). To convert to a positive angle, add 360 (or 2π or 400) to the angle.

For the second vector entered, the final keystroke will be *either* Σ+ *or* Σ−, depending on whether the two vectors should be added or subtracted.

Section 5

The Display and Continuous Memory

Display Control

The HP-15C has three display formats—[FIX], [SCI], and [ENG]—that use a given number (0 through 9) to specify display format. The illustration below shows how the number 123,456 would be displayed specified to four places in each possible mode.

[f][FIX] 4 : **123,456.0000**
[f][SCI] 4 : **1.2346 05**
[f][ENG] 4 : **123.46 03**

Owing to Continuous Memory, any change you make in the display format will be preserved until Continuous Memory is reset.

The current display format takes effect when digit entry is terminated; until then, all digits you key in (up to 10) are displayed.

Fixed Decimal Display

[FIX] (*fixed decimal*) format displays a figure with the number of decimal places you specify (up to nine, depending on the size of the integer portion.) Exponents will be displayed if the number is too small or too large for the display. At "power-up," the HP-15C is in [FIX] 4 format. The key sequence is [f][FIX] *n*.

Keystrokes	Display	
123.4567895	**123.4567895**	
[f][FIX] 4	**123.4568**	
[f][FIX] 6	**123.456790**	Display is rounded to six decimal places. (Ten places are stored internally.)
[f][FIX] 4	**123.4568**	Usual [FIX] 4 display.

Scientific Notation Display

[SCI] (*scientific*) format displays a number in scientific notation. The sequence [f][SCI] *n* specifies the number of decimal places to be shown. Up to six decimal places can be shown since the exponent

display takes three spaces. The display will be rounded to the specified number of decimal places; however, if you specify more decimal places than the six places the display can hold (that is, [SCI] 7, 8, or 9), rounding will occur in the *undisplayed* seventh, eighth, or ninth decimal place.*

With the previous number still in the display:

Keystrokes	Display		
[f][SCI]6	**1.234568**	**02**	Rounds to and shows six decimal places.
[f][SCI]8	**1.234567**	**02**	Rounds to eight decimal places, but displays only six.

Engineering Notation Display

[ENG] (*engineering*) format displays numbers in an engineering notation format in a manner similar to [SCI], except:

- In engineering notation, the first significant digit is always present in the display. The number you key in after [f][ENG] specifies the number of *additional* digits to which you want to round the display.
- Engineering notation shows all exponents in multiples of three.

Keystrokes	Display		
.012345	**0.012345**		
[f][ENG]1	**12.**	**-03**	Rounds to the first digit after the leading digit.
[f][ENG]3	**12.35**	**-03**	
10[×]	**123.5**	**-03**	Decimal shifts to maintain multiple of three in exponent.
[f][FIX]4	**0.1235**		Usual [FIX] 4 format.

* Therefore, the display shows no distinction among [SCI] 7, 8, and 9 *unless* the number rounded up is a 9, which carries a 1 over into the next higher decimal place.

Mantissa Display

Regardless of the display format, the HP-15C always internally holds each number as a 10-digit mantissa and a two-digit exponent of 10. For example, π is always represented internally as $3.141592654 \times 10^{00}$, regardless of what is in the display.

When you want to view the full 10-digit mantissa of a number in the X-register, press [f] CLEAR [PREFIX]. To keep the mantissa in the display, hold the [PREFIX] key down.

Keystrokes	Display
[g] [π]	**3.1416**
[f] CLEAR [PREFIX]	
(hold)	**3141592654**

Round-Off Error

As mentioned earlier, the HP-15C holds every value to 10 digits internally. It also rounds the final result of every calculation to the 10th digit. Because the calculator can provide only a finite approximation for numbers such as π or 2/3 (0.666...), a small error due to rounding can occur. This error can be increased in lengthy calculations, but usually is insignificant. To accurately assess this effect for a given calculation requires numerical analysis beyond our scope and space here! Refer to the *HP-15C Advanced Functions Handbook* for a more detailed discussion.

Special Displays

Annunciators

The HP-15C display contains eight annunciators that indicate the status of the calculator for various operations. The meaning and use of these annunciators is discussed on the following pages:

*	Low-power indication, page 62.
USER	User mode, pages 79 and 144.
f and g	Prefixes for alternate functions, pages 18–19.
RAD and **GRAD**	Trigonometric modes, page 26.
C	Complex mode, page 121.
PRGM	Program mode, page 66.

Digit Separators

The HP-15C is set at power-up so that it separates integral and fractional portions of a number with a period (a decimal point), and separates groups of three digits in the integer portion with a comma. You can reverse this setting to conform to the numerical convention used in many countries. To do so, turn off the calculator. Press and hold [ON], press and hold [.], release [ON], then release [.] ([ON]/[.]). (Repeating this sequence will set the calculator to the previous display convention.)

Keystrokes	Display
12345.67	**12,345.67**
[ON]/[.]	**12.345,6700**
[ON]/[.]	**12,345.6700**

Error Display

If you attempt an improper operation—such as division by zero—an error message (**Error** followed by a digit) will appear in the display. For a complete listing of error messages and their causes, refer to appendix A.

To clear the **Error** display and restore the calculator to its prior condition, press any key. You can then resume normal operation.

Overflow and Underflow

Overflow. When the result of a calculation in *any* register is a number with a magnitude greater than $9.999999999 \times 10^{99}$, $\pm 9.999999999 \times 10^{99}$ is placed in the affected register and the overflow flag, flag 9, is set.* Flag 9 causes the display to blink. When overflow occurs in a running program, execution continues until completion of the program, and then the display blinks.

The blinking can be stopped and flag 9 cleared by pressing [←], [ON], or [g][CF]9.

Underflow. If the result of a calculation in any register is a number with a magnitude less than $1.000000000 \times 10^{-99}$, that number will be replaced by zero. Underflow does not have any other effect.

---

* Recall that the display does not include the last three digits of the mantissa.

Low-Power Indication

When a flashing asterisk, which indicates low battery power, appears in the lower left-hand side of the display, there is no reason to panic. You still have plenty of calculator time remaining: at least 10 minutes if you continuously run programs, and at least an hour if you do calculations manually. Refer to appendix F (page 259) for information on replacing the batteries.

0.0000
*

Continuous Memory

Status

The Continuous Memory feature of the HP-15C retains the following in the calculator, even when the display is turned off:

- All numeric data stored in the calculator.
- All programs stored in the calculator.
- Position of the calculator in program memory.
- Display mode and setting.
- Trigonometric mode (Degrees, Radians, or Grads).
- Any pending subroutine returns.
- Flag settings (except flag 9, which clears when the display is *manually* turned off).
- User mode setting.
- Complex mode setting.

When the HP-15C is turned on, it always "wakes up" in Run mode.

If the calculator is turned off, Continuous Memory will be preserved for a short period while the batteries are removed. Data and programs are preserved longer than other aspects of calculator status. Refer to appendix F for instructions on changing batteries.

Resetting Continuous Memory

If at any time you want to reset (entirely clear) the HP-15C Continuous Memory:

1. Turn the calculator off.
2. Press and hold the [ON] key, then press and hold the [−] key.
3. Release the [ON] key, then the [−] key. (This convention is represented as [ON] / [−].)

When Continuous Memory is reset, **Pr Error** (*power error*) will be displayed. Press any key to clear the display.

Note: Continuous Memory can inadvertently be interrupted and reset if the calculator is dropped or otherwise traumatized.

Part II
HP-15C
Programming

Section 6

Programming Basics

The next five sections are dedicated to explaining aspects of programming the HP-15C. Each of these programming sections will first discuss basic techniques (The Mechanics), then give examples for the implementation of these techniques (Examples), and lastly discuss finer points of operation in greater detail (Further Information). Read only as far as you need to support your use of the HP-15C.

The Mechanics

Creating a Program

Programming the HP-15C is an easy matter, based simply on recording the keystroke sequence used when calculating manually. (This is called "keystroke programming".) To create a program out of a series of calculation steps requires two extra manipulations: deciding where and how to enter your data; and loading and storing the program. In addition, programs can be instructed to make decisions and perform iterations through conditional and unconditional branching.

As we step through the fundamentals of programming, we'll rework the falling object program illustrated in the Problem Solver (page 14).

Loading a Program

Program Mode. Press [g] [P/R] (*program/run*) to set the calculator to *Program mode* (**PRGM** annunciator on). Functions are stored and not executed when keys are pressed in Program mode.

Keystrokes	Display	
[g] [P/R]	000-	Switches to Program mode; **PRGM** annunciator and line number (000) displayed.

Location in Program Memory. Program memory—and therefore the calculator's position in program memory—is demarcated by line numbers. Line 000 marks the beginning of program memory and cannot be used to store an instruction. The first line that contains an instruction is line 001. Program lines other than 000 do not exist until instructions are written for them.

You *can* start a program at any existent line (designated *nnn*), but it is simplest and safest to start an independent program (as opposed to a subroutine) at the beginning of program memory. As you write, any existing program lines will be preserved and "bumped" down in program memory.

Press [GTO] [CHS] 000 (in Program or Run mode) to move to line 000 without recording the [GTO] statement. *In Run mode,* [f] CLEAR [PRGM] will also reset the calculator to line 000—without clearing program memory.

Alternatively, you can clear program memory, which will erase all programs in memory and position you to line 000. To do so, press [f] CLEAR [PRGM] *in Program mode.*

Program Begin. A *label* instruction—[f] [LBL] followed by a letter ([A] through [E]) or number (0 through 9 or .0 through .9)—is used to define the beginning of a program or routine. The use of labels allows you to quickly select and run one particular program or routine out of several.

Keystrokes	Display	
[f] CLEAR [PRGM]	000-	Clears program memory and sets to line 000 (start of program memory).
[f] [LBL] [A]	001-42,21,11	

Recording a Program. Any key pressed—operator or constant—will be recorded in memory as a programmed instruction.*

* Except the *nonprogrammable functions,* which are listed on page 80.

Keystrokes	Display		
2	002–	2	Given h in the X-register, lines 002 to 008 calculate $\sqrt{\frac{2h}{9.8}}$.
[×]	003–	20	
9	004–	9	
[÷]	005–	48	
8	006–	8	
[÷]	007–	10	
[√x]	008–	11	

Program End. There are three possible endings for a program:

- [g] [RTN] (*return*) will end a program, return to line 000, and halt.
- [R/S] will stop a program *without* moving to line 000.
- The end of program memory contains an automatic [RTN].

Keystrokes	Display		
[g] [RTN]	009–	43 32	Optional *if this is the last program in memory.*

Intermediate Program Stops

Use [f] [PSE] (*pause*) as a program instruction to *momentarily* stop a program and display an intermediate result. (Use more than one [PSE] for a longer pause.)

Use a [R/S] (*run/stop*) instruction to stop the program indefinitely. The program will remain positioned at that line. You can resume program execution (from that line) by pressing [R/S] during Run mode, that is, from the keyboard.

Running a Program

Run Mode. Switch back to Run mode when you are done programming: [g] [P/R]. Program execution must take place in Run mode.

Keystrokes	Display	
g P/R		Run mode; no **PRGM** annunciator displayed. (The display will depend on any previous result.)

The position in program memory does not change when modes are switched. Should the calculator be shut off, it always "wakes up" in Run mode.

Executing a Program. In *Run mode*, press f *letter label* or GSB *digit* (or letter) *label*. This addresses a program and starts its execution. The display will flash **running**.

Keystrokes	Display	
300.51	**300.51**	Key a value for *h* into the X-register.
f A	**7.8313**	The result of executing program "A". (The number of seconds it takes an object dropped from 300.51 meters high to hit the ground.)

Restarting a Program. Press R/S to continue execution of a program that was stopped with a R/S instruction.

User Mode. User mode is an optional condition to save keystrokes when executing *letter-named* programs. Pressing f USER will interchange the f-shifted and primary functions of the A through E keys. You can then execute a program using just one keystroke (skipping the f or GSB).

How to Enter Data

Every program must take into account how and when data will be supplied. This can be done in Run mode before running the program or during an interruption in the program.

1. **Prior entry.** If a variable value will be used in the first line of the program, enter it into the X-register before starting the program. If it will be used later, you can store it (with STO) into a storage register, and recall it (with a programmed RCL) within the program.

This is the method used above, where *h* was placed in the X-register before running the program. No [ENTER] instruction is necessary because program execution (here: [f] [A]) both terminates digit entry and enables the stack lift. The above program then multiplied the contents of the X-register (*h*) by 2.

The presence of the stack even makes it possible to load more than one variable prior to running a program. Keeping in mind how the stack moves with subsequent calculations and how the stack can be manipulated (as with [x⇄y]), it is possible to write a program to use variables which have been keyed into the X-, Y-, Z-, and T-registers.

2. **Direct entry.** Enter the data as needed as the program runs. Write a [R/S] (*run/stop*) instruction into the program where needed so the program will stop execution. Enter your data, then press [R/S] to restart the program.

Do not key variable data into the program itself. Any values that will vary should be entered anew with each program execution.

Program Memory

At power-up (Continuous Memory reset), the HP-15C offers 322 bytes of program memory and 21 storage registers. *Most* program steps (instructions) use one byte, but some use two. The distribution of memory capacity can be altered, as explained in appendix C. The maximum attainable program memory is 448 bytes (with the permanent storage registers—R_I, R_0, and R_1—remaining); maximum number of storage registers is 67 (with no program memory).

Example

Mother's Kitchen, a canning company, wants to package a ready-to-eat spaghetti mix containing three different cylindrical cans: one of spaghetti sauce, one of grated cheese, and one of meatballs. Mother's needs to calculate the base areas, total surface areas, and volumes of the three different cans. It would also like to know, per package, the total base area, surface area, and volume.

The program to calculate this information uses these formulas and data:

base area = πr^2.
volume = base area × height = $\pi r^2 h$.
surface area = 2 base areas + side area = $2\pi r^2 + 2\pi rh$.

Radius, r	Height, h	Base Area	Volume	Surface Area
2.5 cm	8.0 cm	?	?	?
4.0	10.5	?	?	?
4.5	4.0	?	?	?
	TOTALS	?	?	?

Method:

1. Enter an r value into the calculator and save it for other calculations. Calculate the base area (πr^2), store it for later use, and add the base area to a register which will hold the sum of all base areas.
2. Enter h and calculate the volume ($\pi r^2 h$). Add it to a register to hold the sum of all volumes.
3. Recall r. Divide the volume by r and multiply by 2 to yield the side area. Recall the base area, multiply by 2, and add to the side area to yield the surface area. Sum the surface areas in a register.

Do *not* enter the actual data while writing the program—just *provide for* their entry. These values will vary and so will be entered before and/or during each program run.

Key in the following program to solve the above problem. The display shows line numbers and keycodes (the row and column location of a key), which will be explained under Further Information.

Keystrokes	Display	
g P/R	000–	Sets calculator to Program mode (**PRGM** displayed).
f CLEAR PRGM	000–	Clears program memory. Starts at line 000.

Keystrokes	Display	
f LBL A	001-42,21,11	Assigns this program the label "A".
STO 0	002- 44 0	Stores the contents of X-register into R_0. r must be in the X-register before running the program.
g x^2	003- 43 11	Squares the contents of the X-register (which will be r).
g π	004- 43 26	
×	005- 20	πr^2, the BASE AREA of a can.
STO 4	006- 44 4	Stores the BASE AREA in R_4.
STO + 1	007-44,40, 1	Keeps a sum of all BASE AREAS in R_1.
R/S	008- 31	Stops to display BASE AREA and allow entry of the h value.
×	009- 20	Multiplies h by the BASE AREA, giving VOLUME.
f PSE	010- 42 31	Pauses briefly to display VOLUME.
STO + 2	011-44,40, 2	Keeps a sum of all can VOLUMES in R_2.
RCL 0	012- 45 0	Recalls r.
÷	013- 10	Divides VOLUME by r.
2	014- 2	
×	015- 20	$2\pi rh$, the SIDE AREA of a can.
RCL 4	016- 45 4	Recalls the BASE AREA of the can.
2	017- 2	Multiplies base area by two (for top and bottom).
×	018- 20	

Keystrokes	Display	
[+]	019- 40	SIDE AREA + BASE AREA = SURFACE AREA.
[STO] [+] 3	020-44,40, 3	Keeps a sum of all SURFACE AREAS in R_3.
[g] [RTN]	021- 43 32	Ends the program and returns program memory to line 000.

Now, let's run the program:

Keystrokes	Display	
[g] [P/R]		Sets calculator to Run mode. (**PRGM** cleared.)
[f] CLEAR [REG]		Clears *all* storage registers. The display does not change.
2.5	2.5	Enter *r* of the first can.
[f] [A] (or: [GSB] [A])	19.6350	Starts program A. BASE AREA of first can. (**running** flashes during execution.)
8	8	Enter *h* of first can. Then restart program.
[R/S]	157.0796	VOLUME of first can.
	164.9336	SURFACE AREA of first can.
4	4	Enter *r* of the second can.
[R/S]	50.2655	BASE AREA of second can.
10.5	10.5	Enter *h* of second can.
[R/S]	527.7876	VOLUME of second can.
	364.4247	SURFACE AREA of second can.
4.5	4.5	Enter *r* of the third can.
[R/S]	63.6173	BASE AREA of third can.

Keystrokes	Display	
4	**4**	Enter *h* of third can.
R/S	**254.4690**	VOLUME of third can.
	240.3318	SURFACE AREA of third can.
RCL 1	**133.5177**	Sum of BASE AREAS.
RCL 2	**939.3362**	Sum of VOLUMES.
RCL 3	**769.6902**	Sum of SURFACE AREAS.

The preceding program illustrates the basic techniques of programming. It also shows how data can be manipulated in Program and Run modes by entering, storing, and recalling data (input and output) using ENTER, STO, RCL, storage register arithmetic, and programmed stops.

Further Information

Program Instructions

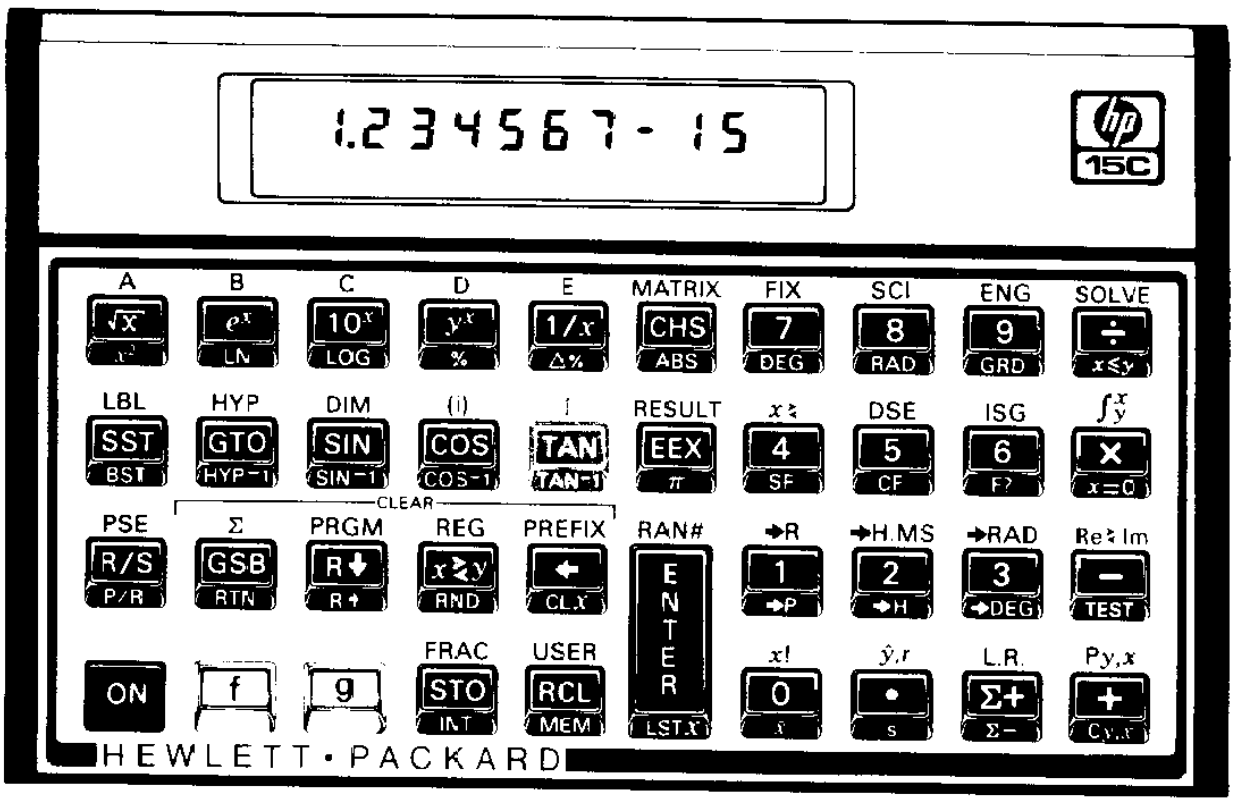
Each digit, decimal point, and function key is considered an *instruction* and is stored in one *line* of program memory. An instruction may include prefixes (such as f, STO, GTO, and LBL) and still occupy only one line. Most instructions require one *byte* of program memory; however, some require two. For a complete list of two-byte instructions, refer to appendix C.

Instruction Coding

Each key on the HP-15C keyboard—except for the digit keys 0 through 9—is identified in Program mode by a two-digit "keycode" that corresponds to the key's position on the keyboard.

Instruction	Code	
STO + 1	**006-44,40, 1**	Sixth program line.
f DSE I	**XXX-42, 5,25**	DSE is just "5".

The first digit of a keycode refers to the row (1 to 4 from top to bottom), and the second digit refers to the column (1, 2, ... 9, 0 from left to right). Exception: the keycode for a digit key is simply that digit.



Key only 25 second [illegible] fifth key

Memory Configuration

Understanding memory configuration is not essential to your use of the HP-15C. It is essential, however, for obtaining maximum efficiency in memory and programming use. The more you program, the more useful this knowledge will be. Memory configuration and allocation is thoroughly explained in appendix C, Memory Allocation.

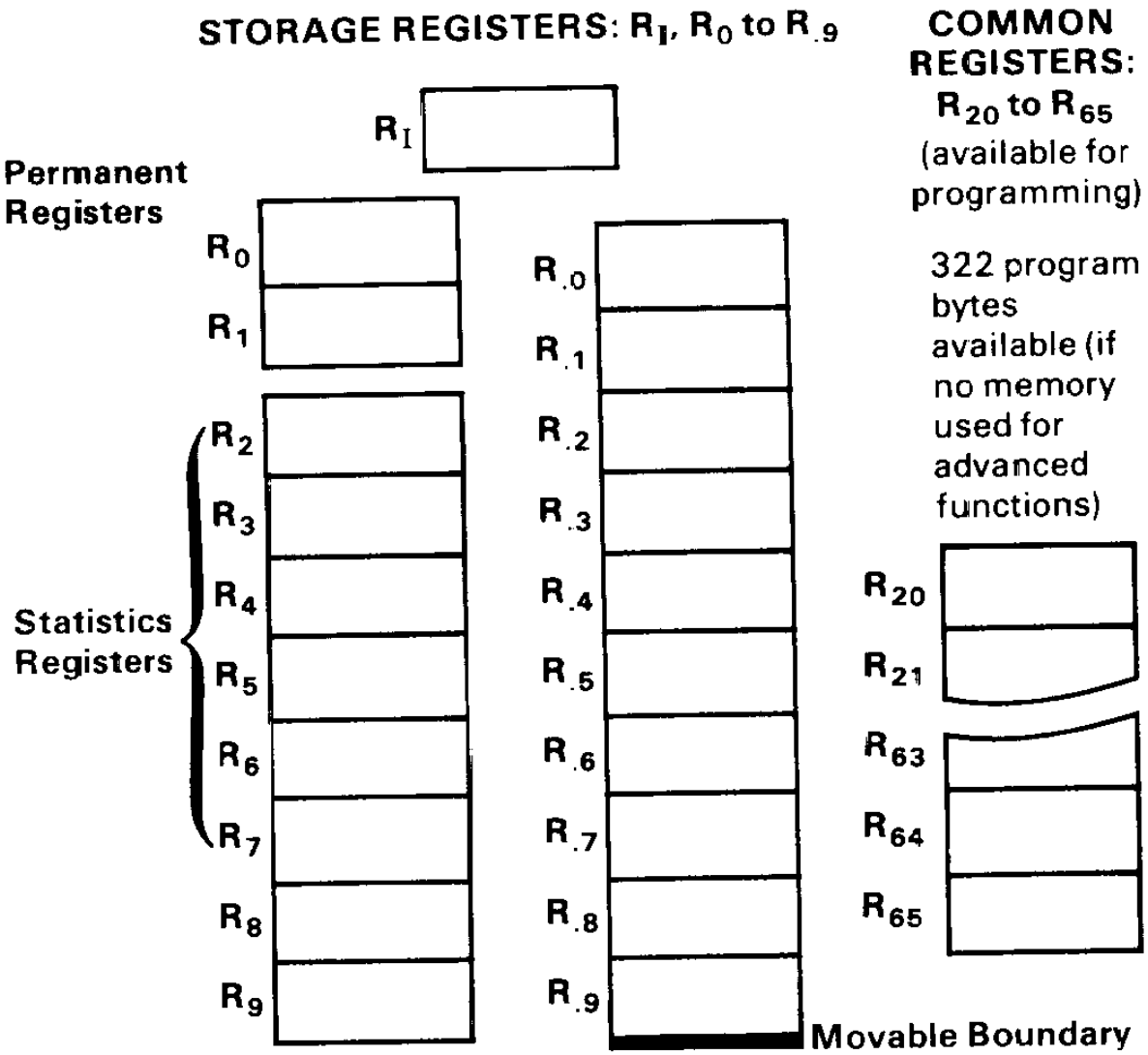
Should you ever get an **Error 10**, you have run up against limitations of the HP-15C memory. If you learn how to reallocate memory, you can greatly increase your ability to store information in the HP-15C.

The HP-15C memory consists of 67 registers (R_0 to R_{65} and the Index register) divided between data storage and programming/advanced function capability. The initial configuration is:

- 46 registers for both programming and the advanced functions (SOLVE, f, the imaginary stack, and MATRIX functions). At seven bytes of memory per register, this is worth 322 program bytes if no memory is dedicated to advanced functions.

- 21 registers for data storage (R_0 to R_9, $R_{.0}$ to $R_{.9}$, and the Index register).

Initial Memory Configuration



Memory is reallocated by telling the calculator which data storage register shall be the highest data register; all other registers are left for programming and advanced functions.

Keystrokes	Display	
60 [f] [DIM] [(i)]*	**60.0000**	R_{60} and below allocated to data storage; five (R_{61} to R_{65}) remain for programming.

* The optional omission of the [f] keystroke *after* another prefix key is explained on page 78, Abbreviated Key Sequences.

Keystrokes	Display	
1 [f] [DIM] [(i)]	**1.0000**	R_1 and R_0 allocated for data storage; R_2 to R_{65} available for programming and advanced functions.
19 [f] [DIM] [(i)]	**19.0000**	Original allocation: R_{19} ($R_{.9}$) and below for data storage; R_{20} to R_{65} for programming and advanced functions.*
[RCL] [DIM] [(i)]	**19.0000**	Displays the current highest data register.

The [DIM] and [MEM] (*memory status*) functions are described in detail in appendix C.

Keep in mind that an error message will result (*given the above memory configuration*) if

1. You try to address a register higher than R_{19} ($R_{.9}$), which initially is the highest register allocated to data storage (**Error 3**).
2. You have 322 occupied program bytes and try to load more program lines (**Error 4**).
3. You try to run an advanced function with insufficient available memory (**Error 10**).

Program Boundaries

End. Not every program needs to end with a [RTN] or [R/S] instruction. If you are at the end of occupied program memory, there is an *automatic* [RTN] instruction, so you do not need to enter one. This can save you one line of memory. On the other hand, a program can "end" by simply transferring execution to another routine using [GTO] (section 7).

Labels. Labels in a program (or subroutine) are markers telling the calculator where to begin execution. Following an [f] *label* or [GSB] *label* instruction, the calculator will search downward in

---

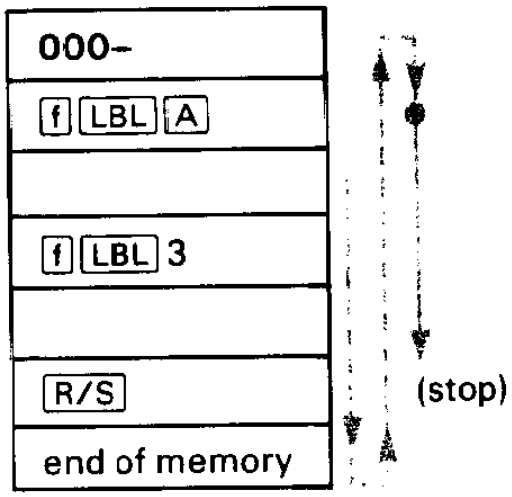
* For memory allocation and indirect addressing, registers $R_{.0}$ through $R_{.9}$ are referred to as R_{10} through R_{19}.

program memory for the corresponding label. If need be, the search will wrap around at the end of program memory and continue at line 000. When it encounters an appropriate label, the search stops and execution begins.

If a label is encountered as part of a running program, it has no effect, that is, execution simply continues. Therefore, you can label a subordinate routine within a program (more on subroutines in section 9).

Since the calculator searches in only one direction from its present position, it is possible (though not advisable) to use duplicate program labels. Execution will begin at the first appropriately labeled line encountered.

If an [f] [A] entry starts the search for "A" here,



it then proceeds downward through memory, wraps around to line 000, and stops at label "A". Execution then starts and continues (ignoring any other labels) until a halt instruction.

Unexpected Program Stops

Pressing Any Key. Pressing any key will halt program execution. It will *not* halt in the middle of an operation. This instruction will be completed before the program stops.

Error Stops. Program execution is immediately halted when the calculator attempts an improper operation that results in an **Error** display.

To see the line number and keycode of the error-causing instruction (the line at which the program stopped), press any one key to remove the **Error** message, then switch to Program mode.

If the display is flashing when a program stops, an overflow condition exists (page 61). Press [←], [ON], or [g] [CF] 9 to stop the blinking.

Abbreviated Key Sequences

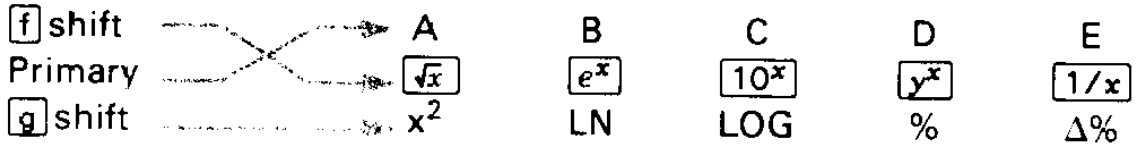
In certain cases, an [f] prefix you might expect to include in a key

sequence is not needed. The rule for using an *abbreviated key sequence* is: the [f] prefix key is unnecessary after any other prefix key. (Page 19 contains a list of prefix keys.)

For example, [f] [LBL] [f] [A] becomes [f] [LBL] [A], [f] [DIM] [f] [(i)] becomes [f] [DIM] [(i)], and [STO] [f] [RAN#] becomes [STO] [RAN#]. The removal of the [f] is not ambiguous because the [f]-shifted function is the only logical one in these cases. The keycodes for such instructions do not include the extraneous [f] even if you do key it in.

User Mode

User mode is a convenience to save keystrokes when addressing (calling up) programs for execution. Pressing [f] [USER] will exchange the primary functions and [f]-shifted functions of the [A] through [E] keys only. In User mode (**USER** annunciator displayed):



Press [f] [USER] again to deactivate User mode.

Polynomial Expressions and Horner's Method

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

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

uses the variable x four different times. A program to solve such an equation could repeatedly recall a stored copy of x from a storage register. A shorter programming method, however, would be to use a stack which has been filled with the constant (refer to Loading the Stack with a Constant, page 41).

Horner's Method is a useful means of rearranging polynomial expressions to cut calculation steps and calculation time. It is especially expedient in [SOLVE] and [∫], two rather long-running functions that use subroutines.

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

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

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

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

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

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

Keystrokes	Display		
g P/R	000-		Assumes position in memory is line 000. If it is not, clear program memory.
f LBL B	001-42,21,12		
5	002-	5	
×	003-	20	$5x$.
2	004-	2	
+	005-	40	$5x + 2$.
×	006-	20	$(5x + 2)x$.
×	007-	20	$(5x + 2)x^2$.
×	008-	20	$(5x + 2)x^3$.
g RTN	009-	43 32	
g P/R			Returns to Run mode. Prior result remains in display.
7 ENTER ENTER ENTER	7.0000		Loads the stack (X-, Y-, Z-, and T-registers) with 7.
f B	12,691.0000		

Nonprogrammable Functions

When the calculator is in Program mode, almost every function on the keyboard can be recorded as an instruction in program memory. The following functions *cannot* be stored as instructions in program memory.

f CLEAR PREFIX	g BST	SST
f CLEAR PRGM	g MEM	←
f (i)	g P/R	ON / ·
f USER	GTO CHS *nnn*	ON / −

Problems

1. The village of Sonance has installed a 12-o'clock whistle in the firehouse steeple. The sound level at the firehouse door, 3.2 meters from the whistle, is 138 decibels. Write a program to find the sound level at various distances from the whistle.

   Use the equation $L = L_0 - 20 \log (r/r_0)$, where:
   L_0 is the known sound level (138 db) at a point near the source,
   r_0 is the distance of that point from the source (3.2 m),
   L is the unknown sound level at a second point, and
   r is the distance of the second point from the source in meters.

   What is the sound level at 3 km from the source ($r = 3$ km)?

   A possible keystroke sequence is:
   [g] [P/R] [f] [LBL] [C] 3.2 [÷] [g] [LOG] 20 [×] [CHS] 138 [+] [g] [RTN] [g] [P/R] taking 15 program lines and 15 bytes of memory. This problem can be solved in a more general way by removing the specific values 3.2 and 138 from the program, and instead recalling the L_0 and r_0 values from storage registers; or by removing 3.2 and 138 and loading L_0, r, and r_0 into the stack before execution: L_0 [ENTER] r [ENTER] r_0.

   (Answer: for $r = 3$ km, $L = 78.5606$ db.)

2. A "typical" large tomato weighs about 200 grams, of which about 188 g (94%) are water. A tomato grower is trying to produce tomatoes of lower percentage water. Write a program to calculate the percent change in water content of a given tomato compared to the typical tomato. Use a programmed stop to enter the water weight of the new tomato.

   What is the percent change in water content for a 230 g tomato of which 205 g are water?

   A possible keystroke sequence is:
   [f] [LBL] [D] .94 [ENTER] [R/S] (enter water weight of new tomato) [ENTER] [R/S] (enter weight of new tomato) [÷] [g] [Δ%] [g] [RTN] taking 11 program lines and 11 bytes of memory.

   (Answer: for the 230 g tomato above, the percent change in percent water weight is −5.1804%.)

Section 7

Program Editing

There are many reasons to modify a program after you've already stored it: you might want to add or delete an instruction (like [STO], [PSE], or [R/S]), or you might even find some errors! The HP-15C is equipped with several editing features to make this process as easy as possible.

The Mechanics

Making a program modification of any kind involves two steps: moving to the proper line (the location of the needed change) and making the deletion(s) and/or insertion(s).

Moving to a Line in Program Memory

The Go To ([GTO]) Instruction. The sequence [GTO] [CHS] *nnn* will move program memory to line number *nnn*, whether pressed in Run mode or Program mode (**PRGM** displayed). This is *not* a programmable sequence; it is for *manually* finding a specific position in program memory. The line number must be a three-digit number satisfying $000 \leqslant nnn \leqslant 448$.

The Single Step ([SST]) Instruction. To move only one line at a time forward through program memory, press [SST] (*single step*). This function is not programmable.

In Program mode: [SST] will move the memory position forward one line and display that instruction. The instruction is *not* executed. If you hold the key down, the calculator will continuously scroll through the lines in program memory.

In Run mode: [SST] will display the current program line while the key is held down. When the key is released, the current instruction is executed, the result displayed, and the calculator steps forward to the next program line to be executed.

The Back Step ([BST]) Instruction. To move one line *backwards* in program memory, press [BST] (*back step*) in Program or Run mode. This function is not programmable. [BST] will scroll (with the key held down) in Program mode. Program instructions are *not* executed.

Deleting Program Lines

Deletions of program instructions are made with [←] (*back arrow*) *in Program mode*. Move to the line you want to delete, then press [←]. Any remaining following lines will be renumbered to stay in sequence.

Pressing [←] in Run mode does not affect program memory, but is used for display clearing. (Refer to page 21.)

Inserting Program Lines

Additions to a program are made by moving to the line *preceding* the point of insertion. Any instruction you key in will be added *following* the line currently in the display. To alter an instruction, first delete it, then add the new version.

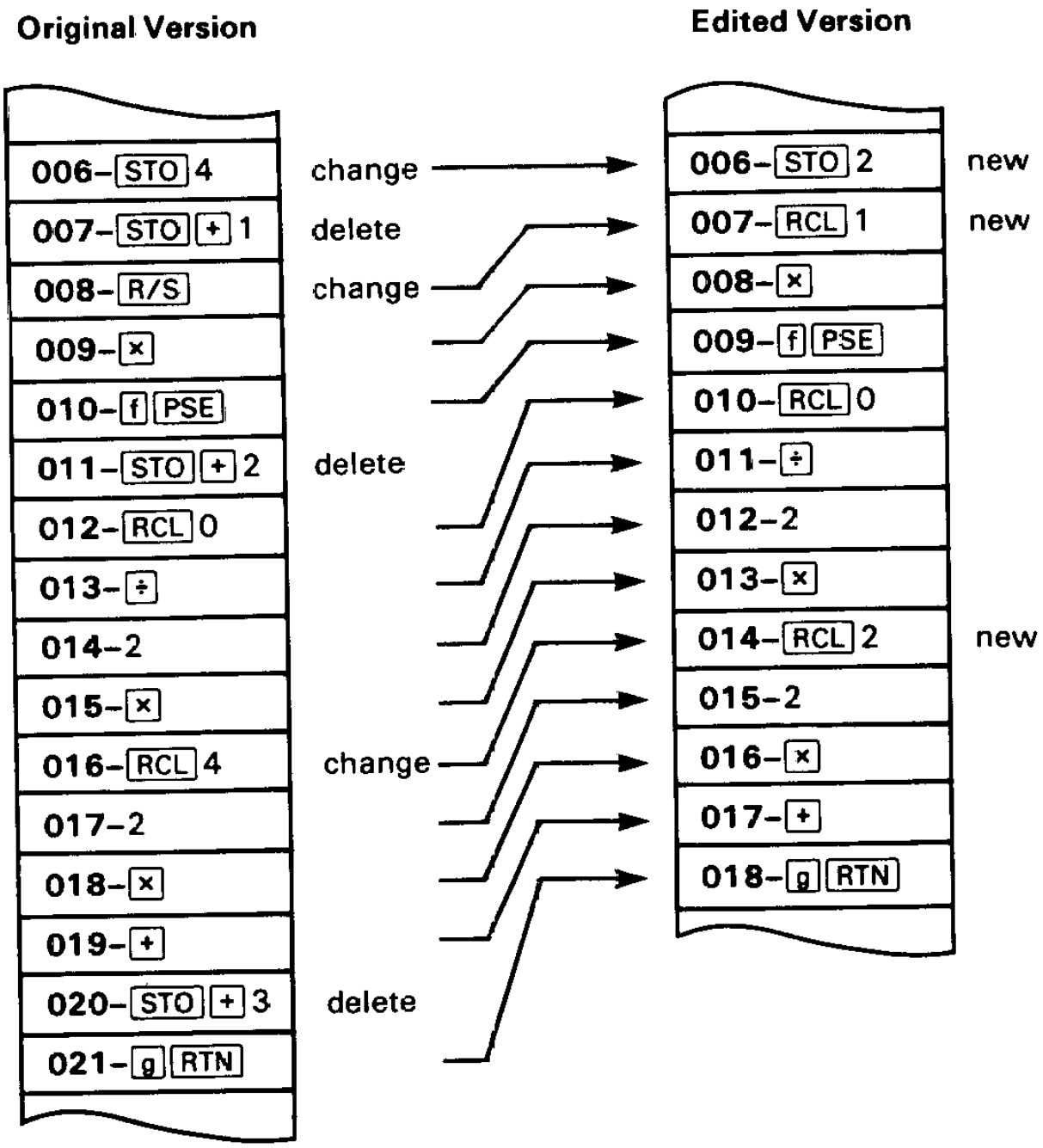
Examples

Let's refer back to the can volume program on page 71 in section 6 and make a few changes in the instructions. (The can program as listed below is assumed to be in memory starting on line 001.)

Deletions: If we don't need the summed base area, volume, and surface area values, we can delete the storage register additions (lines 007, 011, and 020).

Changes: To eliminate the need to stop the program to enter the height value (h), change the [R/S] instruction to a [RCL] 1 instruction (because of the above deletions, R_1 is no longer being used) and store h in R_1 before running the program. To clean things up, let's also alter [STO] 4 (line 006) to [STO] 2 and [RCL] 4 (old line 016) to [RCL] 2, since we are no longer using R_2 and R_3.

The editing process is diagrammed on the next page.

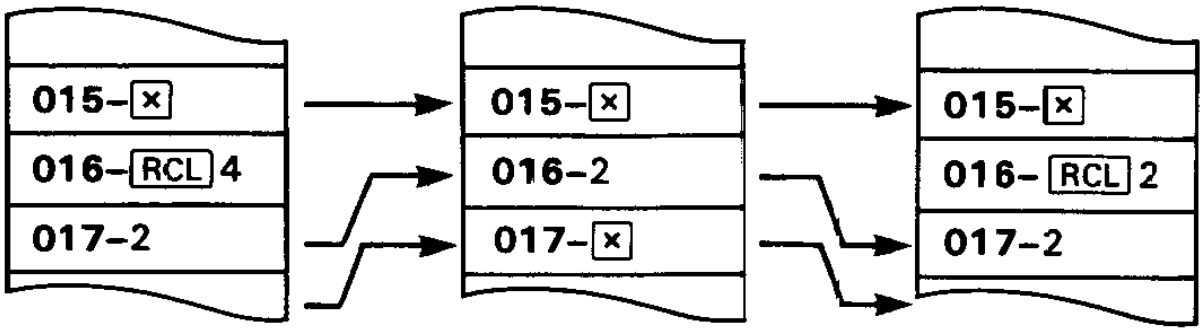


Let's start at the end of the program and work backwards. In this way, deletions will not change the line numbers of the preceding lines in the program.

Keystrokes	Display	
g P/R	000-	Program mode. (Assumes position is at line 000.)
GTO CHS 020 (or use SST)	020-44,40, 3	Moves position to line 020 (instruction STO + 3.)

Keystrokes	Display	
[←]	019- 40	Line 020 deleted.
[g] [BST] (hold)	016- 45 4	The next line to edit is line 016 ([RCL] 4).
[←]	015- 20	Line 016 deleted.
[RCL] 2	016- 45 2	Line 016 changed to [RCL] 2.
[GTO] [CHS] 011 (or hold [BST])	011-44,40, 2	Moves to line 011 ([STO] [+] 2).
[←]	010- 42 31	Line 011 deleted.
[g] [BST] (hold)	008- 31	Stop! (Single-stepping backwards to line 008: [R/S].)
[←]	007-44,40, 1	[R/S] deleted.
[RCL] 1	008- 45 1	Line 008 changed to [RCL] 1.
[g] [BST]	007-44,40, 1	Back-step to line 007.
[←]	006- 44 4	Line 007 ([STO] [+] 1) deleted.
[←]	005- 20	Line 006 ([STO] 4) deleted.
[STO] 2	006- 44 2	Changed to [STO] 2.

The replacement of a line proceeds like this:



Further Information

Single-Step Operations

Single-Step Program Execution. If you want to check the contents of a program or the location of an instruction, you can single step through the program *in Program mode*. If, on the other

hand, running the program produces an error, or you suspect that a portion of the program is faulty, you can check the program by *executing* it stepwise. This is done by pressing [SST] *in Run mode.*

Keystrokes	Display	
[g] [P/R]		Run mode.
[f] CLEAR [REG]		Clear storage registers.
[GTO] [A]		Move to first line of program A.
8 [STO] 1	**8.0000**	Store a can height.
2.5	**2.5**	Enter a can radius.
[SST] (hold)	**001-42,21,11**	Keycode for line 001 (label).
(release)	**2.5000**	Result of executing line 001.
[SST]	**002- 44 0**	[STO] 0.
	2.5000	Result.
[SST]	**003- 43 11**	[g] [x^2].
	6.2500	Result.
[SST]	**004- 43 26**	[g] [π].
	3.1416	Result.
[SST]	**005- 20**	[×].
	19.6350	Result: the base area of the can.

Wrapping. [SST] will not move program position into "unoccupied" program territory. Instead, the calculator will "wrap around" to line 000. (In Run mode, [SST] will perform any instructions at the end of program memory, such as [RTN], [GTO], or [GSB].)

Line Position

Recall that the calculator's position in program memory does not change when it is shut off or Program/Run modes are changed. Upon returning to Program mode, the calculator line position will be where you left it. (If you executed a program ending with [RTN], the position will be at line 000.) Therefore, if the calculator is left on and shuts itself off, you need only turn it on and switch to Program mode (the calculator always "wakes up" in Run mode) to be back where you were.

Insertions and Deletions

After an insertion, the display will show the instruction you just added. After a deletion, the display will show the line prior to the deleted (now nonexistent) one.

If all space available in memory is occupied, the calculator will not accept any program instruction insertions and **Error 4** will be displayed.

Initializing Calculator Status

The contents of storage registers and the status of calculator settings will affect a program if the program uses those registers or depends on a certain status setting. If the current status is incorrect for the program being run, you will get incorrect results. Therefore, it is wise to clear registers and set relevant modes either just prior to running a program or within the program itself. A self-initializing program is more mistake-proof—but it also uses more program lines.

Calculator-initializing functions are: [f] CLEAR [Σ], [f] CLEAR [PRGM], [f] CLEAR [REG], [g] [DEG], [g] [RAD], [g] [GRD], [g] [SF], and [g] [CF].

Problems

It is good programming technique to avoid using identical program labels. (This shouldn't be hard, since the HP-15C provides 25 different labels.) To ensure against duplication of labels, you can clear program memory first.

1. The following program is used by the manager of a savings and loan company to compute the future values of savings accounts according to the formula $FV = PV\,(1 + i)^n$, where FV is future value, PV is present value, i is the periodic interest rate, and n is the number of periods. Enter PV first (into the Y-register) and n second (into the X-register) before executing the program. Given is an annual interest rate of 7.5% (so $i = 0.075$).

Keystrokes	Display	
f LBL . 1	001-42,21, .1	
f FIX 2	002-42, 7, 2	
1	003- 1	} Interest.
.	004- 48	
0	005- 0	
7	006- 7	
5	007- 5	
x≷y	008- 34	
y^x	009- 14	$(1+i)^n$.
×	010- 20	$PV(1+i)^n$.
g RTN	011- 43 32	

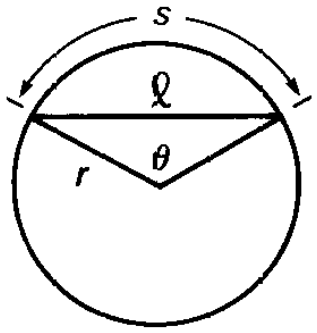
Load the program and find the future value of \$1,000 invested for 5 years; of \$2,300 invested for 4 years. Remember to use GSB to run a program with a digit label. (Answers: \$1,435.63; \$3,071.58.)

Alter the program to make the annual interest rate 8.0%.

Using the edited program, find the future value of \$500 invested for 4 years; of \$2,000 invested for 10 years. (Answers: \$680.24; \$4,317.85.)

2. Create a program to calculate the length of a chord ℓ subtended by an angle θ (in degrees) on a circle of radius r, according to the equation

$$\ell = 2r \sin \frac{\theta}{2}.$$



Find ℓ when $\theta = 30°$ and $r = 25$.

(Answer: 12.9410. A possible program is: f LBL A g DEG f FIX 4 2 × x≷y 2 ÷ SIN × g RTN). (Assumes θ in Y-register and r in X-register when program is run.)

Make any necessary modifications in the program to also find and display s, the length of the circular arc cut by θ (*in radians*), according to the equation

$$s = r\,\theta.$$

Complete the following table:

θ	r	ℓ	s
45°	50	?	?
90°	100	?	?
270°	100	?	?

(Answers: 38.2683 and 39.2699; 141.4214 and 157.0796; 141.4214 and 471.2389. A possible new sequence is: [f] [LBL] [A] [g] [DEG] [f] [FIX] 4 [STO] 0 2 [×] [$x \gtrless y$] [STO] 1 2 [÷] [SIN] [×] [f] [PSE] [f] [PSE] [RCL] 0 [RCL] 1 [f] [→RAD] [×] [g] [RTN]).

Section 8

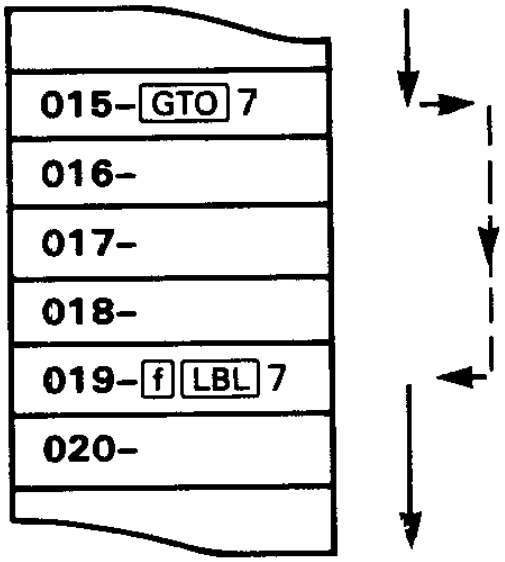
Program Branching and Controls

Although the instructions in a program are normally executed sequentially, it is often desirable to transfer execution to a part of the program *other than* the next line. *Branching* in the HP-15C may be *simple*, or it may depend on a certain *condition*. By branching to a previous line, it is possible to execute part of a program more than once—a process called *looping*.

The Mechanics

Branching

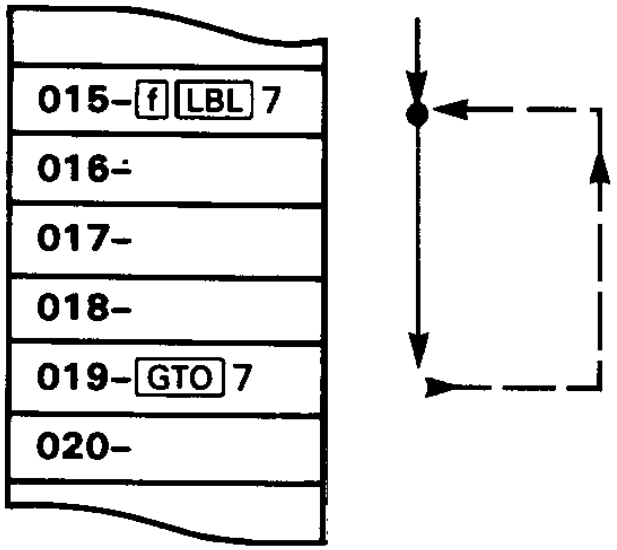
The Go To ([GTO]) Instruction. Simple branching—that is, unconditional branching—is carried out with the instruction [GTO] *label*. In a running program, [GTO] will transfer execution to the next appropriately labeled program or routine (*not* to a line *number*).



The calculator searches forward in memory, wrapping around through line 000 if necessary, and resumes execution at the first line containing the proper label.

Looping. If a [GTO] instruction specifies a label at a lower-numbered line (that is, a prior line), the series of instructions

between the [GTO] and the label will be executed repeatedly—possibly indefinitely. The continuation of this loop can be controlled by a conditional branch, an [R/S] instruction (written into the loop), or simply by pressing any key during execution (which stops the program).



Conditional Tests

Another way to alter the sequence of program execution is by a *conditional test*, a true/false test which compares the number in the X-register either to zero or to the number in the Y-register. The HP-15C provides 12 different tests, two explicit on the keyboard and 10 others accessible using [g] [TEST] *n*.*

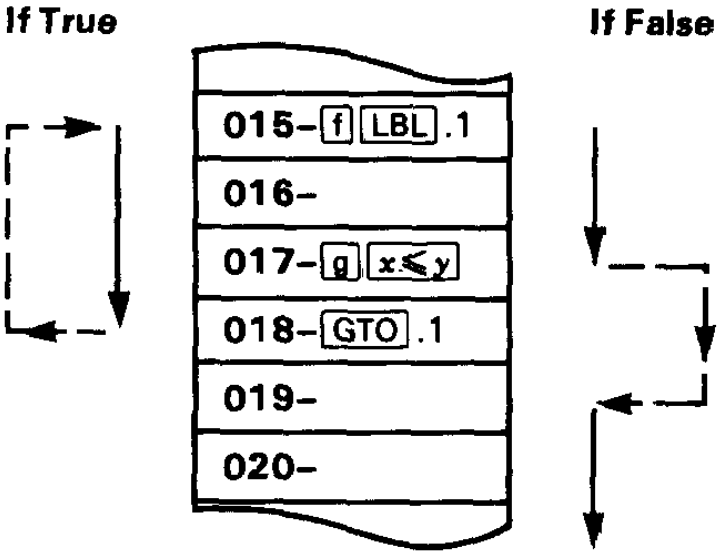
1. Direct: [g] [x≤y] and [g] [x=0].
2. Indirect: [g] [TEST] *n*.

n	Test	*n*	Test
0	$x \neq 0$	5	$x = y$
1	$x > 0$	6	$x \neq y$
2	$x < 0$	7	$x > y$
3	$x \geqslant 0$	8	$x < y$
4	$x \leqslant 0$	9	$x \geqslant y$

* Four of the conditional tests can also be used for complex values, as explained in section 11 on page 132.

Following a conditional test, program execution follows the "Do if True" Rule: it proceeds sequentially if the condition is true, and it *skips one instruction if the condition is false*. A [GTO] instruction is often placed right after a conditional test, making it a *conditional branch*; that is, the [GTO] branch is executed only if the test condition is met.

Program Execution After Test



Flags

Another conditional test for programming is a *flag* test. A flag is a status indicator that is either *set* (= true) or *clear* (= false). Again, execution follows the "Do if True" Rule: it proceeds sequentially if the flag is set, and skips one line if the flag is clear.

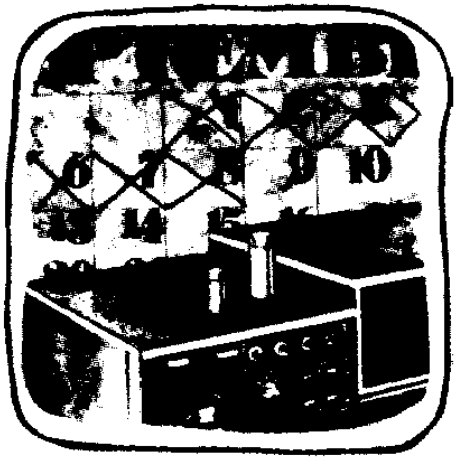
The HP-15C has eight *user* flags, numbered 0 to 7, and two *system* flags, numbered 8 (Complex mode) and 9 (overflow condition). The system flags are discussed later in this section. All flags can be set, cleared, and tested as follows:

- [g][SF] *n* will *set flag* number *n* (0 to 9).
- [g][CF] *n* will *clear flag* number *n*.
- [g][F?] *n* will *check* if flag *n* is set.

A flag *n* that has been set remains set until it is cleared either by a [CF] *n* instruction or by clearing (resetting) Continuous Memory.

Examples

Example: Branching and Looping



A radiobiology lab wants to predict the diminishing radioactivity of a test amount of ^{131}I, a radioisotope. Write a program to figure the radioactivity at 3-day intervals until a given limit is reached. The formula for N_t, the amount of radioisotope remaining after t days, is

$$N_t = N_0 (2^{-t/k}),$$

where $k = 8$ days, the half-life of ^{131}I, and N_0 is the initial amount.

The following program uses a loop to calculate the number of millicuries (mci) of isotope theoretically remaining at 3-day intervals of decay. Included is a conditional test to check the result and end the program when radioactivity has fallen to a given value (a limit).

The program assumes t_1—the first day of measurement—is stored in R_0, N_0—the initial amount of isotope—is stored in R_1, and the limit value for radioactivity is stored in R_2.

Keystrokes	Display		Comments
g P/R	000-		Program mode.
f CLEAR PRGM	000-		(Optional.)
f LBL A	001-	42,21,11	Each loop returns to this line.
RCL 0	002-	45 0	Recalls current t, which changes with each loop.
f PSE	003-	42 31	Pauses to display t.
8	004-	8	k.
÷	005-	10	
CHS	006-	16	$-t/k$.
2	007-	2	
x≷y	008-	34	
y^x	009-	14	$2^{-t/k}$.

Keystrokes	Display	
RCL × 1	010–45,20, 1	Recall multiplication with the contents of R_1 (N_0), yielding N_t, the mci of ^{131}I remaining after t days.
f PSE	011– 42 31	Pauses to display N_t.
RCL 2	012– 45 2	Recalls limit value to X-register.
g TEST 9	013–43,30, 9	$x \geq y$? Tests whether limit value (in X) meets or exceeds N_t (in Y).
g RTN	014– 43 32	If so, program ends.
3	015– 3	If not, program continues.
STO + 0	016–44,40, 0	Adds 3 days to t in R_0.
GTO A	017– 22 11	Go to "A" and repeat execution to find a new N_t from a new t.

Notice that without lines 012 to 014, the loop would run indefinitely (until stopped from the keyboard).

Let's run the program, using $t_1 = 2$ days, $N_0 = 100$ mci, and a limit value of half of N_0 (50 mci).

Keystrokes	Display	
g P/R		Run mode (display will vary).
2 STO 0	2.0000	t_1.
100 STO 1	100.0000	N_0.
50 STO 2	50.0000	Limit value for N_t.
f A	2.0000	t_1.
	84.0896	N_1.
	5.0000	t_2.
	64.8420	N_2.
	8.0000	t_3.
	50.0000	N_3.
	50.0000	N_t limit; program ends.

Example: Flags

Calculations on debts or investments can be calculated in two ways: for payments made in advance (at the beginning of a given period) and for payments made in arrears (at the end of a given period). If you write a program to calculate the value (or "present value") of a debt or investment with periodic interest and periodic payments, you can use a flag as a status indicator to tell the program whether to assume payments are made in advance or payments are made in arrears.

Suppose you are planning the payment of your child's future college tuition. You expect the cost to be about $3,000/year or about $250/month. If you wanted to withdraw the monthly payments from a bank account yielding 6% per year, compounded monthly (which equals 0.5% per month), how much must you deposit in the account at the start of the college years to fund monthly payments for the next 4 years?

The formula is

$$V = P\left[\frac{1-(1+i)^{-n}}{i}\right](1+i)$$

if payments are to be made each month in advance,

and the formula is

$$V = P\left[\frac{1-(1+i)^{-n}}{i}\right]$$

if payments are to be made each month in arrears.

V is the total value of the deposit you must make in the account;

P is the size of the *periodic* payment you will draw from the account;

i is the *periodic* interest rate (here: "periodic" means monthly, since interest is compounded monthly); and

n is the number of compounding periods (months).

The following program allows for either payment mode. It assumes that, before the program is run, P is in the Z-register, n is in the Y-register, and i is in the X-register.

Keystrokes	Display	
g P/R	000-	Program mode.
f LBL B	001-42,21,12	Start at "B" if payments to be made at the beginning.
g CF 0	002-43, 5, 0	Flag 0 clear (false); indicates advance payments.
GTO 1	003- 22 1	Go to main routine.
f LBL E	004-42,21,15	Start at "E" if payments to be made at the end.
g SF 0	005-43, 4, 0	Flag 0 set (true); indicates payment in arrears.
f LBL 1	006-42,21, 1	Routine 1 (main routine).
STO 1	007- 44 1	Stores i (from X-register).
1	008- 1	
+	009- 40	$(1+i)$.
x≷y	010- 34	Puts n in X; $(1+i)$ in Y.
CHS	011- 16	$-n$.
y^x	012- 14	$(1+i)^{-n}$.
CHS	013- 16	$-(1+i)^{-n}$.
1	014- 1	
+	015- 40	$1-(1+i)^{-n}$.
RCL ÷ 1	016-45,10, 1	Recall division with R_1 (i) to get $[1-(1+i)^{-n}]/i$.
×	017- 20	Multiplies quantity by P.
g F? 0	018-43, 6, 0	Flag 0 set?
g RTN	019- 43 32	End of calculation if flag 0 set (for payments in arrears).
RCL 1	020- 45 1	Recalls i.
1	021- 1	
+	022- 40	$(1+i)$.
×	023- 20	Multiplies quantity by final term.
g RTN	024- 43 32	End of calculation if flag 0 clear.

Now run the program to find the total amount needed in an account from which you want to take $250/month for 48 months. Enter the periodic interest rate as a decimal fraction, that is, 0.005 per month. First find the sum needed if payments will be made at the beginning of the month (payments in advance), then calculate the sum needed if payments will be made at the end of the month (in arrears).

Keystrokes	Display	
g P/R		Set to Run mode.
250 ENTER	**250.0000**	Monthly payment.
48 ENTER	**48.0000**	Payment periods (4 years × 12 months).
.005	**0.005**	Monthly interest rate as a decimal fraction.
f B	**10,698.3049**	Deposit necessary for payments to be made in advance.
(Repeat stack entries.)		
f E	**10,645.0795**	Deposit necessary for payments to be made in arrears. (The difference between this deposit and the tuition cost ($12,000) represents interest earned on the deposit!)

Further Information

Go To

In contrast to the nonprogrammable sequence GTO CHS *nnn*, the programmable sequence GTO *label* cannot be used to branch to a line *number*, but only to a *program label* (a line containing f LBL *label*).* Execution continues from the point of the new label, and does not return to the original routine unless given another GTO instruction.

* It is possible to branch under program control to a particular line *number* by using indirect addressing, discussed in section 10.

[GTO] *label* can also be used in Run mode (that is, from the keyboard) to move to a labeled position in program memory. No execution occurs.

Looping

Looping is an application of branching which uses a [GTO] instruction to repeat a portion of the program. A loop can continue indefinitely, or may be conditional. A loop is frequently used to repeat a calculation with different variables. At the same time, a counter, which increments with each loop, may be included to keep track of loop iterations. This counter can then be checked with a conditional test to determine when to exit the loop. (This is shown in the example on page 112.)

Conditional Branching

There are two general applications for conditional branching. One is to control loops, as explained above. A conditional test can check for either a certain calculated value or a certain loop count.

The other major use is to test for options and pursue one. For example, if a salesperson made a variable commission depending on the amount of sale, you could write a program which takes the amount of sale, compares it to a test value, and then calculates a specific commission depending on whether the sale is less than or greater than the test value.

Tests. A conditional test takes what is in the X-register ("x") and compares it either to zero (such as [x=0]) or to "y", that is, what is in the Y-register (such as [x≤y]). For an x:y comparison, therefore, you must have the x- and y-values juxtaposed in the X- and Y-registers. This might require that you store a test value and then recall it (bringing it into the X-register). Or, the value might be in the stack and be moved, as necessary, using [x≷y], [R↓], or [R↑].

Tests With Complex Numbers and Matrix Descriptors. Four of the conditional tests also work with complex numbers and matrix descriptors: [x=0], [TEST] 0 ($x \neq 0$), [TEST] 5 ($x = y$), and [TEST] 6 ($x \neq y$). Refer to sections 11 and 12 for more information.

Flags

As a conditional test can be used to pick an option by comparing two numbers in a program, a flag can be used to pick an option

externally. Usually, a flag is set or cleared first thing in a program by choosing a different starting point (using different labels) depending on the condition or mode you want (refer to the example on page 95).

In this way, a program can accommodate two different modes of input, such as degrees and radians, and make the correct calculation for the mode chosen. You set a flag if a conversion needs to be made, for instance, and clear it if no conversion is needed.

Suppose you had an equation requiring temperature input in degrees Kelvin, although sometimes your data might be in degrees Celsius. You could use a program with a flag to allow either a Kelvin or Celsius input. In part, such a program might include:

f LBL C	Start program at "C" for degrees Celsius.
g CF 7	Flag 7 cleared (=false).
GTO 1	
f LBL D	Start program at "D" for degrees Kelvin.
g SF 7	Flag 7 set (=true).
f LBL 1	(Assuming temperature in X-register.)
g F? 7	Checks for flag 7 (checks for Celsius or Kelvin input).
GTO 2	If set (Kelvin input), goes to a later routine, skipping the next few instructions.
2	If cleared (Celsius input), adds 273 to the
7	value in the X-register, since °K = °C + 273.
3	
+	
f LBL 2	Calculation continues for both modes.
⋮	

The System Flags: Flags 8 and 9

Flag 8. Setting flag 8 will activate Complex mode (described in section 11), turning on the **C** annunciator. If another method is used to activate Complex mode, flag 8 will automatically be set. Complex mode is deactivated only by clearing flag 8; flag 8 is cleared in the same manner as the other flags.

Flag 9. An overflow condition (described on page 61) automatically sets flag 9. Flag 9 causes the display to blink or, if a program is running, waits until execution is complete and then starts blinking the display.

Flag 9 may be cleared in three ways:

- Press [g] [CF] 9 (the common procedure for clearing flags).
- Press [←]. This will only clear flag 9 and stop the blinking—it will not clear the display.
- Turn the calculator off. (Flag 9 is not cleared if the calculator turns itself off.)

If you set flag 9 manually ([SF] 9), it causes the display to blink irrespective of the overflow status of the calculator. As usual, a program will run to completion before the display starts blinking. Therefore, flag 9 can be used as a programming tool to provide a visual signal for a selected condition.

Section 9

Subroutines

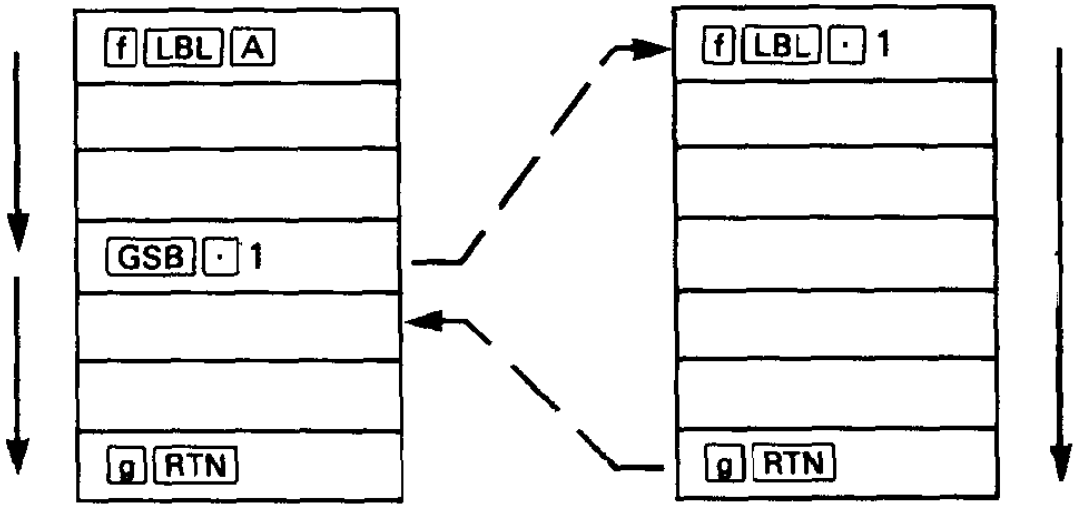
When the same set of instructions needs to be used at more than one point in a program, memory space can be conserved by storing those instructions as a single subroutine.

The Mechanics

Go To Subroutine and Return

The [GSB] (*go to subroutine*) instruction is executed in the same way as the [GTO] branch, with one major difference: it establishes a *pending return* condition. [GSB] *label*, like [GTO] *label*,* transfers program execution to the line with the corresponding label ([A] to [E], 0 to 9 or .0 to .9). However, execution then continues *until the first subsequent* [RTN] *instruction is encountered*—at which point execution *transfers back* to the instruction immediately following the last [GSB] instruction, and continues on from there.

Subroutine Execution



END

Execution transfers to line 000 and halts.

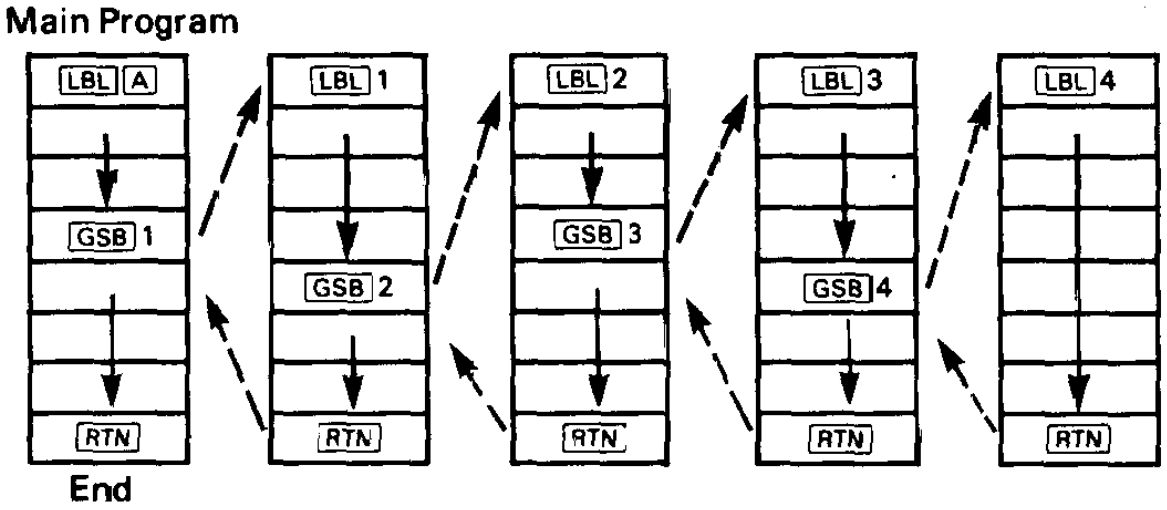
RETURN

Execution transfers back to original routine, *after* [GSB] [.] 1.

* A [GSB] or [GTO] instruction followed by a *letter* label is an abbreviated key sequence (no [f] necessary). Abbreviated key sequences are explained on page 78.

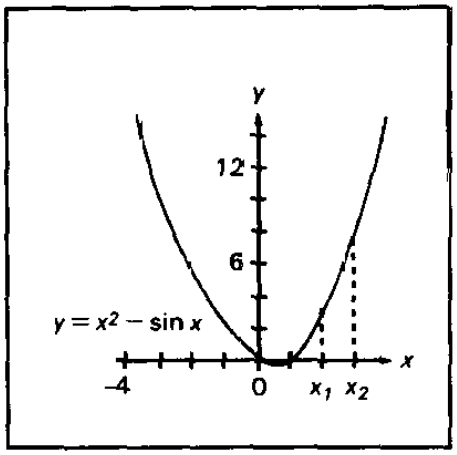
Subroutine Limits

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



Examples

Example: Write a program to calculate the slope of the secant line joining points (x_1, y_1) and (x_2, y_2) on the graph shown, where $y = x^2 - \sin x$ (given x in radians).

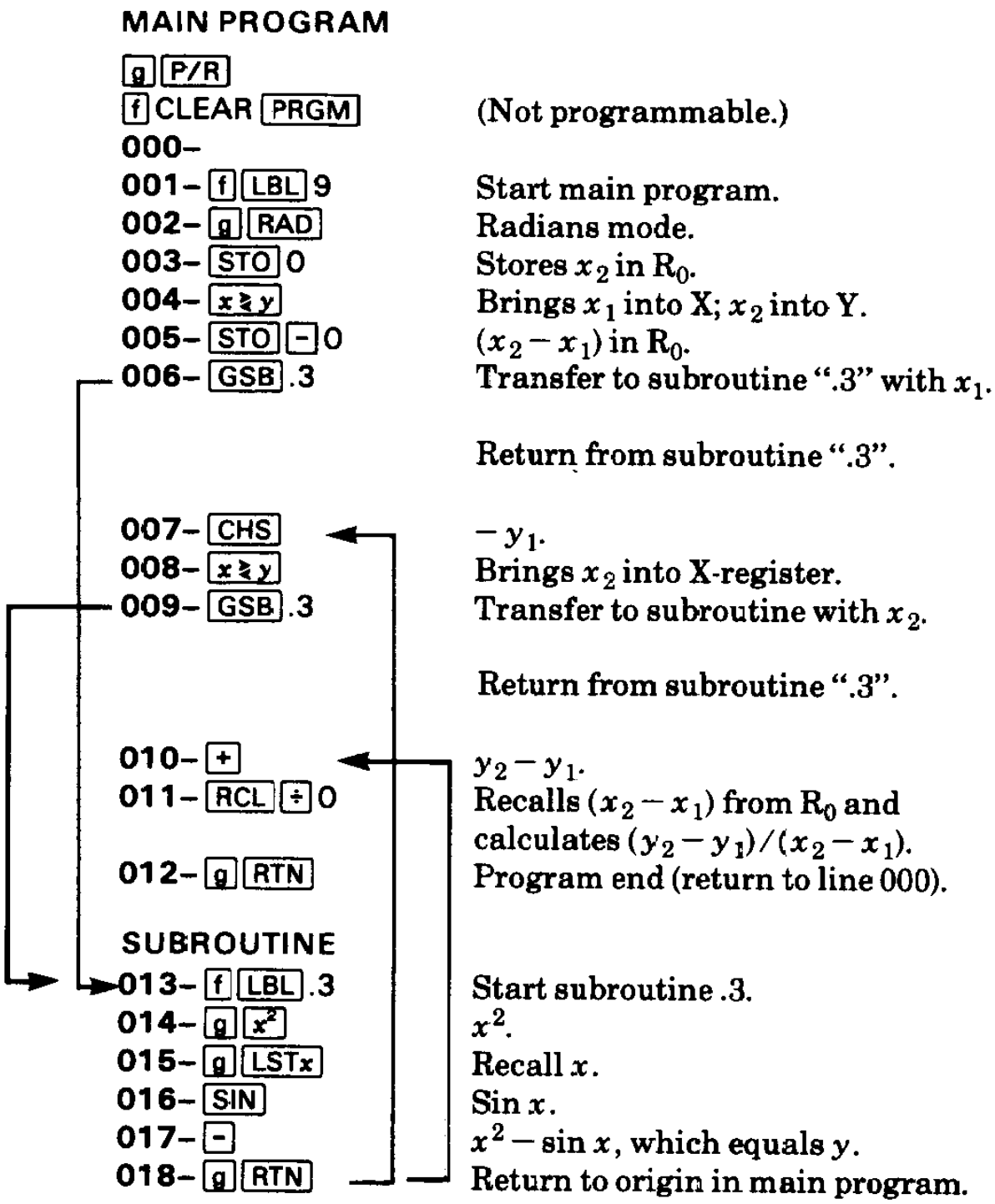


The secant slope is:

$$\frac{y_2 - y_1}{x_2 - x_1}, \text{ or } \frac{(x_2^2 - \sin x_2) - (x_1^2 - \sin x_1)}{x_2 - x_1}$$

The solution requires that the equation for y be evaluated twice—once for y_1 and once for y_2, given the data input for x_1 and x_2. Since the same calculation must be made for different values, it will save program space to call a subroutine to calculate y.

The following program assumes that x_1 has been entered into the Y-register and x_2 into the X-register.



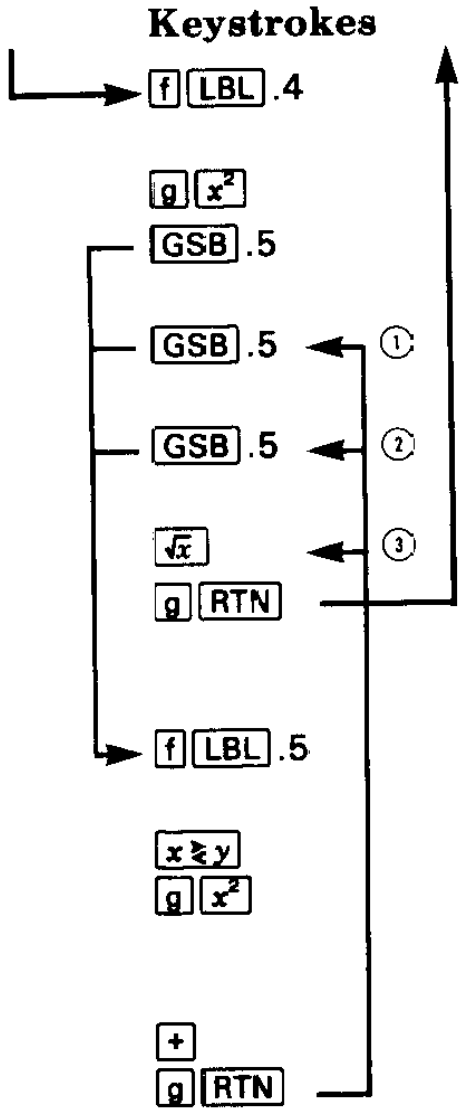
Calculate the slope for the following values of x_1 and x_2: 0.52, 1.25; −1, 1; 0.81, 0.98. Remember to use GSB 9 (rather than f 9) when addressing a routine with a digit label.

Answers: 1.1507; −0.8415; 1.1652.

Example: Nesting. The following subroutine, labeled ".4", calculates the value of the expression $\sqrt{x^2 + y^2 + z^2 + t^2}$ as part of a larger calculation in a larger program. The subroutine calls upon

another subroutine (a nested subroutine), labeled ".5", to do the repetitive squaring.

The program is executed after placing the variables t, z, y, and x into the T-, Z-, Y-, and X-registers.



Start of main subroutine.

x^2.

Calculates y^2 and x^2+y^2.

Calculates z^2 and $x^2+y^2+z^2$.

Calculates t^2 and $x^2+y^2+z^2+t^2$.

$\sqrt{x^2+y^2+z^2+t^2}$

End of main subroutine; returns to main program.

Start of nested subroutine.

Calculates a square and adds it to current sum of squares.

End of nested subroutine; returns to main subroutine.

If you run the subroutine (with its nested subroutine) alone using $x = 4.3$, $y = 7.9$, $z = 1.3$, and $t = 8.0$, the answer you get upon pressing GSB .4 is 12.1074.

Further Information

The Subroutine Return

The *pending return* condition means that the [RTN] instruction occurring subsequent to a [GSB] instruction causes a return to the line following the [GSB] *rather than* a return to line 000. This is what makes a subroutine useful and reuseable in different parts of a program: it will always return execution to where it branched from, even as that point changes. The only difference between using a [GSB] branch and a [GTO] branch is the transfer of execution *after* a [RTN].

Nested Subroutines

If you attempt to call a subroutine that is nested more than seven levels deep, the calculator will halt and display **Error 5** when it encounters the [GSB] instruction at the eighth level.

Note that there is no limitation (other than memory size) on the number of nonnested subroutines or *sets* of nested subroutines that you may use.

Section 10

The Index Register and Loop Control

The Index register (R_I) is a powerful tool in advanced programming of the HP-15C. In addition to storage and recall of data the Index register can use an index number to:

- Count and control loops.
- Indirectly address storage registers, including those beyond $R_{.9}$ (R_{19}).
- Indirectly branch to program line *numbers*, as well as to labels.
- Indirectly control the display format.
- Indirectly control flag operations.

The [I] and [(i)] Keys

Direct Versus Indirect Data Storage With the Index Register

The Index register is a data storage register that can be used directly, with [I], or indirectly, with [(i)].* The difference is important to note:

[I]

The [I] function uses the *number itself* in the Index register.

[(i)]

The [(i)] function uses the absolute value of the integer portion of the number in the Index register to address another data storage register. This is called *indirect addressing*.

* Note that the matrix functions and complex functions use the [I] and [(i)] keys also, but for different purposes. Refer to sections 11 and 12 for their usage.

Indirect Program Control With the Index Register

The [I] key is used for all forms of indirect program control *other than* indirect register addressing. Hence, [I] (not [(i)]) is used for indirect program branching, indirect display format control, and indirect flag control.

Program Loop Control

Program loop counting and control can be carried out in the HP-15C *by any storage register*: R_0 through R_9, $R_{.0}$ through $R_{.9}$, or the Index register ([I]). Loop control can also be carried out *indirectly* with [(i)].

The Mechanics

Both [I] and [(i)] can be used in abbreviated key sequences, omitting the preceding [f] prefix (as explained on page 78).

Index Register Storage and Recall

Direct. [STO] [I] and [RCL] [I]. Storage and recall between the X-register and the Index register operate in the same manner as with other data storage registers (page 42).

Indirect. [STO] (or [RCL]) [(i)] stores into (or recalls from) the data storage register whose number is addressed by the integer portion of the value (0 to 65) in the Index register. See the table below and on the next page.

Indirect Addressing

If R_I contains:	[(i)] will address:	[GTO] [I] or [GSB] [I] will transfer to:*
± 0	R_0	[f] [LBL] 0
⋮	⋮	⋮
9	R_9	[f] [LBL] 9
10	$R_{.0}$	" " .0
11	$R_{.1}$	" " .1
⋮	⋮	⋮
19	$R_{.9}$	[f] [LBL] .9
20	R_{20}	" " [A]

* For $R_I \geq 0$ only.

(Continued on next page.)

Indirect Addressing

If R_I contains:	(i) will address:	GTO I or GSB I will transfer to:*
21	R_{21}	f LBL B
22	R_{22}	" " C
23	R_{23}	" " D
24	R_{24}	" " E
⋮	⋮	—
65	R_{65}	—

* For $R_I \geq 0$ only.

Index Register Arithmetic

Direct. STO or RCL { +, −, ×, ÷ } I. Storage or recall arithmetic operates with the Index register in the same manner as upon other data storage registers (page 43).

Indirect. STO or RCL { +, −, ×, ÷ } (i) carries out storage or recall arithmetic with the contents of the data storage register addressed by the integer portion of the number (0 to 65) in the Index register. See the above table.

Exchanging the X-Register

Direct. f x≷ I exchanges contents between the X-register and the Index register. (Works the same as x≷ *n* does with registers 0 through .9 .)

Indirect. f x≷ (i) exchanges contents between the X-register and the data storage register addressed by the number (0 to 65) in the Index register. See the above table.

Indirect Branching With I

The I key—but *not* the (i) key—can be used for *indirect* branching (GTO I) and subroutine calls (GSB I). (Only the integer portion of the number in R_I is used.) ((i) is *only* used for indirect *addressing* of storage registers.)

To Labels. If the R_I value is *positive*, [GTO] [I] and [GSB] [I] will transfer execution to the *label* which corresponds to the number in the Index register (see the above table).

For instance, if the Index register contains 20.00500, then a [GTO] [I] instruction will transfer program execution to [f] [LBL] [A]. See the chart on page 107.

To Line numbers. If the R_I value is *negative*, [GTO] [I] causes branching to that *line number* (using the absolute value of the integer portion of the value in R_I).

For instance, if R_I contains −20.00500, then a [GTO] [I] instruction will transfer program execution to program line 020.

Indirect Flag Control With [I]

[SF] [I], [CF] [I], or [F?] [I] will set, clear, or test the flag (0 to 9) specified in R_I (by the magnitude of the integer portion).

Indirect Display Format Control With [I]

[f] [FIX] [I], [f] [SCI] [I], and [f] [ENG] [I] will format the display in their customary manner (refer to pages 58-59), using the number in R_I (integer part only) for *n*, which must be from 0 to 9.*

Loop Control With Counters: [ISG] and [DSE]

The [ISG] (*increment and skip if greater than*) and [DSE] (*decrement and skip if less than or equal to*) functions control loop execution by referencing and altering a *loop control number* in a given register. Program execution (skipping a line or not) then depends on that number.

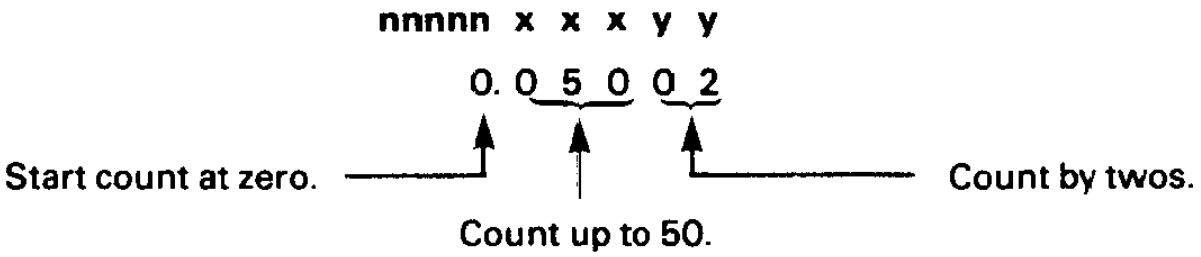
The key sequence is [f] {[ISG], [DSE]} *register number*. This number is 0 to 9, .0 to .9, [I], or [(i)].

The Loop Control Number. The format of the loop control number is:

nnnnn.xxxyy, where	**±nnnnn**	is the current counter value,
	xxx	is the test (goal) value, and
	yy	is the increment or decrement value.

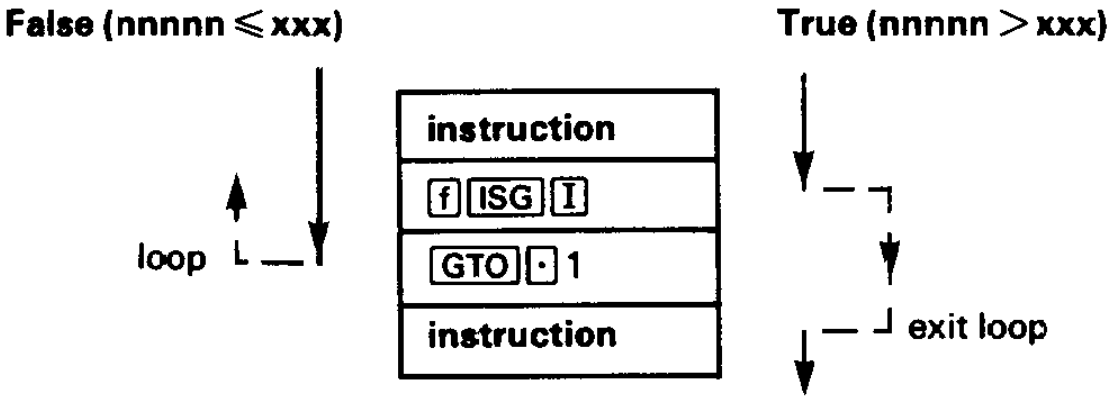
* Except when using [/] (section 14).

For example, the number 0.05002 in a storage register represents:

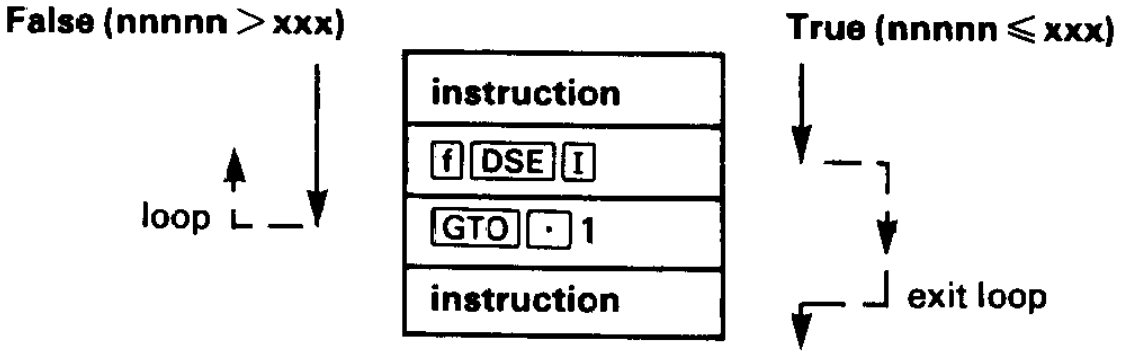


ISG and DSE Operation. Each time a program encounters ISG or DSE, it increments or decrements **nnnnn** (the integer portion of the loop control number), thereby keeping count of loop iterations. It compares **nnnnn** to **xxx**, the prescribed test value, and exits the loop by skipping the next line *if* the loop counter (**nnnnn**) is either greater than (ISG) or less than or equal to (DSE) the test value (**xxx**). The amount that **nnnnn** is incremented or decremented is specified by **yy**.

With these functions (as opposed to the other conditional tests), the rule is "Skip if True".



For ISG: given nnnnn.xxxyy, increment nnnnn to nnnnn + yy, compare it to xxx, and skip the next program line if the new value satisfies nnnnn > xxx. This allows you to exit a loop at this point when nnnnn becomes greater than xxx.



For [DSE]: given **nnnnn.xxxyy**, decrement **nnnnn** to **nnnnn − yy**, compare it to **xxx**, and skip the next program line if the new value satisfies **nnnnn ≤ xxx**. This allows you to exit a loop at this point when **nnnnn** becomes less than or equal to **xxx**.

For example, loop iterations will alter these control numbers as follows:

Iterations

Operation	0	1	2	3	4
ISG	0.00602	2.00602	4.00602	6.00602	8.00602 (skip next line)
DSE	6.00002	4.00002	2.00002	0.00002 (skip next line)	

Examples

Examples: Register Operations

Storing and Recalling

Keystrokes	Display	
f CLEAR REG		Clears all storage registers.
12.3456	12.3456	
STO I	12.3456	Stores in R_I.
7 √x	2.6458	
STO (i)	2.6458	Storage in $R_{.2}$ by indirect addressing ($R_I = 12.3456$).
RCL I	12.3456	Recalls contents of R_I.

Keystrokes	Display	
RCL (i)	**2.6458**	Indirectly recalls contents of $R_{.2}$.
f x≷ .2	**2.6458**	Check: same contents recalled by directly addressing $R_{.2}$.

Exchanging the X-Register

Keystrokes	Display	
f x≷ I	**12.3456**	Exchanges contents of R_I and X-register.
RCL I	**2.6458**	Present contents of R_I.
f x≷ (i)	**0.0000**	Exchanges contents of R_2 (which is zero) with X.
RCL (i)	**2.6458**	
f x≷ 2	**2.6458**	Check: directly address R_2.

Storage Register Arithmetic

Keystrokes	Display	
10 STO + I	**10.0000**	Adds 10 to R_I.
RCL I	**12.6458**	New contents of R_1 (= old + 10).
g π STO ÷ (i)	**3.1416**	Divides contents of $R_{.2}$ by π.
RCL (i)	**0.8422**	New contents of $R_{.2}$.
f x≷ .2	**0.8422**	Check: directly address $R_{.2}$.

Example: Loop Control with DSE

Remember the program in section 8 which used a loop to calculate radioactive decay? (Refer to page 93.) This program used a test condition ($x \geqslant y$?) to exit the loop when the calculated result passed a given limit (50). As we've seen in this section, there's another way to control loop execution: through a stored loop counter that is monitored by the ISG or DSE function.

Here is a revision of the original radioisotope decay program. This time, we will limit the program to three executions of the loop rather than setting a specific limit value. This example uses [DSE] with a loop control number in R_2 of 3. 0 0 0 0 1.

Make the following changes to the program (assuming it is in memory). A loop counter will be stored in R_2 and a line number in the Index register.

Keystrokes	Display	
[g] [P/R]	000–	Program mode.
[GTO] [CHS] 013	013–43,30, 9	The second of the two loop test condition lines.
[←] [←]	011– 42 31	Delete lines 013 and 012.
[f] [DSE] 2	012–42, 5, 2	Add your loop counter function (counter stored in R_2).
[GTO] [I]	013– 22 25	Go to given line number (015).

Now when the loop counter (stored in R_2) has reached zero, it will skip line 013 and go on to 014, the [RTN] instruction, thereby ending the program. If the loop counter has not yet decreased to zero, execution continues with line 013. This branches to line 015 and continues the program and the looping.

To run the program, put t_1 (day 1) in R_0, N_0 (initial isotope batch) in R_1, the loop counter in R_2, and the line number for branching in the Index register.

Keystrokes	Display	
[g] [P/R]		Run mode.
2 [STO] 0	2.0000	t_1.
100 [STO] 1	100.0000	N_0.
3.00001 [STO] 2	3.0000	Loop counter. (This instruction could also be programmed.)

Keystrokes	Display	
15 [CHS] [STO] [I]	**-15.0000**	Branch line number.
[f] [A]	**2.0000**	Running program: loop counter = 3.
	84.0896	
	5.0000	Loop counter = 2.
	64.8420	
	8.0000	Loop counter = 1.
	50.0000	
	50.0000	Loop counter = 0; program ends.

Example: Display Format Control

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.

Keystrokes	
[g] [P/R]	
[f] CLEAR [PRGM]	
[f] [LBL] [B]	
9	**nnnnn** = 9. Therefore, **xxx** = 0 and by default **yy** = 1 (**yy** cannot be zero).
[STO] [I]	
[f] [LBL] 0	
[f] [FIX] [I]	
[RCL] [I]	
[f] [PSE]	Displays current value of **nnnnn**.
[f] [DSE] [I]	**Value in R_I** is decremented and tested. Skip a line if **nnnnn** ≤ test value.
[GTO] 0	Continue loop if **nnnnn** > test value (0).
[g] [TEST] 1	Tests whether **value in display** is greater than 0, so loop will continue when **nnnnn** has reached 0 but display still only shows 1.0.
[GTO] 0	
[g] [RTN]	

To display fixed point notation for all possible decimal places on the HP-15C:

Keystrokes	Display	
g P/R		Run mode.
f B	9.000000000	
	8.00000000	
	7.0000000	
	6.000000	
	5.00000	
	4.0000	
	3.000	
	2.00	
	1.0	
	0.	Display at f PSE instruction.
	0.	Display when program halts.

Further Information

Index Register Contents

Any value stored in the Index register can be referenced in three different ways:

- Using I like any other storage register. The value in R_I can be manipulated as it is: stored, recalled, exchanged, added to, etc.
- Using I as a control number. The absolute value of the integer portion in R_I is a separate entity from the fractional portion. For indirect branching, flag control, and display format control with I, only this portion is used. For loop control, the fractional portion is also used, but separately from the integer portion.*
- Using (i) as a reference to the contents of another storage register. The (i) key uses the indirect addressing system shown in the tables on pages 107 and 108. (In turn, the contents of that second register may be used as a loop control number, in the fashion described above.)

* This is also true for the value in *any* storage register used for indirect loop control.

[ISG] and [DSE]

For the purpose of loop control, the integer portion (the counter value) of the stored control number can be up to five digits long (**nnnnn.xxxyy**). The counter value (**nnnnn**) is zero if not specified otherwise.

xxx, in the decimal portion of the control number, must be specified as a three-digit number. (For example, "5" must be "005".) **xxx** is zero if not specified otherwise. Whenever [ISG] or [DSE] is encountered, **nnnnn** is compared internally to **xxx**, which represents the end level for incrementing or decrementing.

yy must be specified as a two-digit number. **yy** *cannot be zero*, so if left (or specified) as **00**, *the value for* **yy** *defaults to* **1**. The value **nnnnn** is altered by the amount of **yy** each time the loop runs through [ISG] or [DSE]. Both **yy** and **xxx** are reference values, which do *not* change with loop execution.

Indirect Display Control

While you can use the Index register to format the display manually (that is, from the keyboard), this function is most commonly used in programming. This capability is especially valuable for the [F] function, for which accuracy can be stipulated by specifying the number of digits to be displayed (as described in section 14).

There are, as usual, certain display limitations to keep in mind. Recall that any display format function merely alters the number of decimal places to which the display is rounded. In its memory, the calculator always retains a number in scientific notation as a 10-digit mantissa with a two-digit exponent.

The integer portion of the number in the Index register specifies the number of decimal places to which the display is rounded. A number less than zero defaults to zero (zero decimal places displayed in [FIX] format), while a number greater than 9 defaults to 9 (9 decimal places displayed in [FIX]).*

---

*Note that in [SCI] and [ENG] format modes, the maximum display is a seven-digit mantissa with a two-digit exponent. However, a format number greater than six (and less than or equal to nine) *will* alter the decimal place at which rounding occurs. (Refer to pages 58-59.)

An exception is in the case of [f], where the display format number in R_I may range from −6 to +9. (This is discussed in appendix E on page 247.) A number less than zero will not affect the display format, but will affect accuracy with this function.