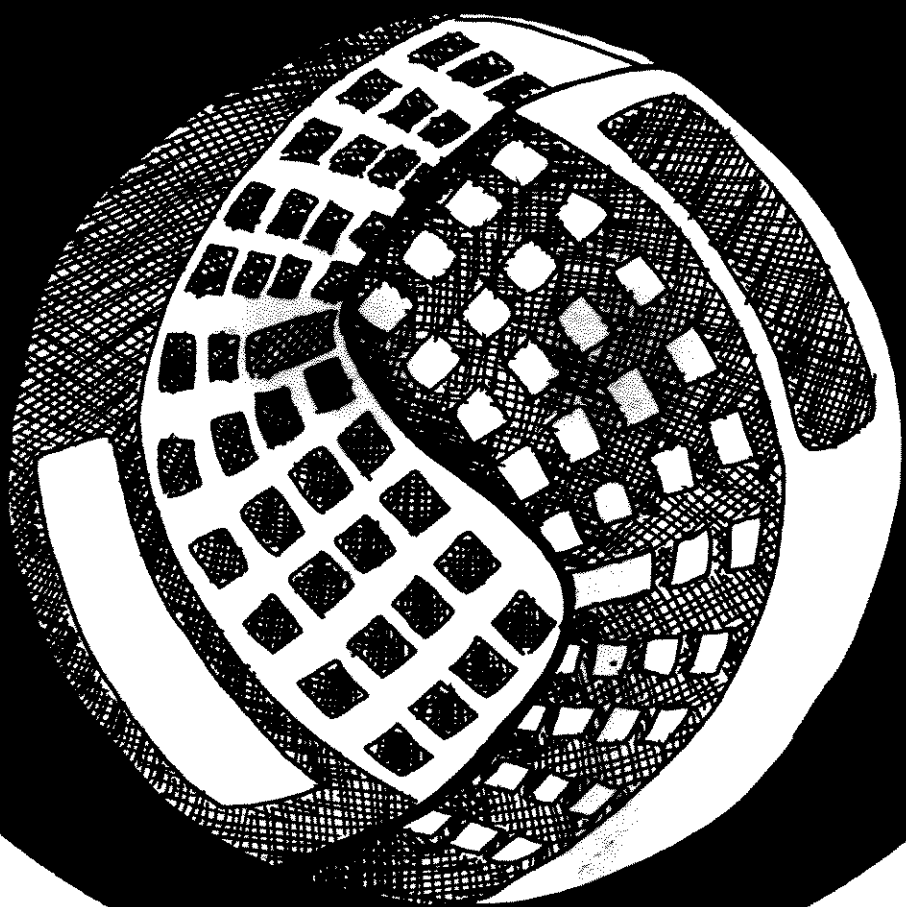
An Easy Course In Programming The



HP-11C and HP-15C

Ted Wadman and Chris Coffin
Illustrated by Robert Bloch

AN EASY COURSE IN PROGRAMMING THE HP–11C AND HP–15C

by Ted Wadman and Chris Coffin
Illustrated by Robert Bloch

Additional editing by Gregg Kleiner and Soraya Simons

Grapevine Publications, Inc.
P.O. Box 118
Corvallis, Oregon 97339–0118

We extend our thanks to Hewlett-Packard Company for producing such top-quality products and documentation.

Printed in the United States of America

ISBN 0-931011-02-7

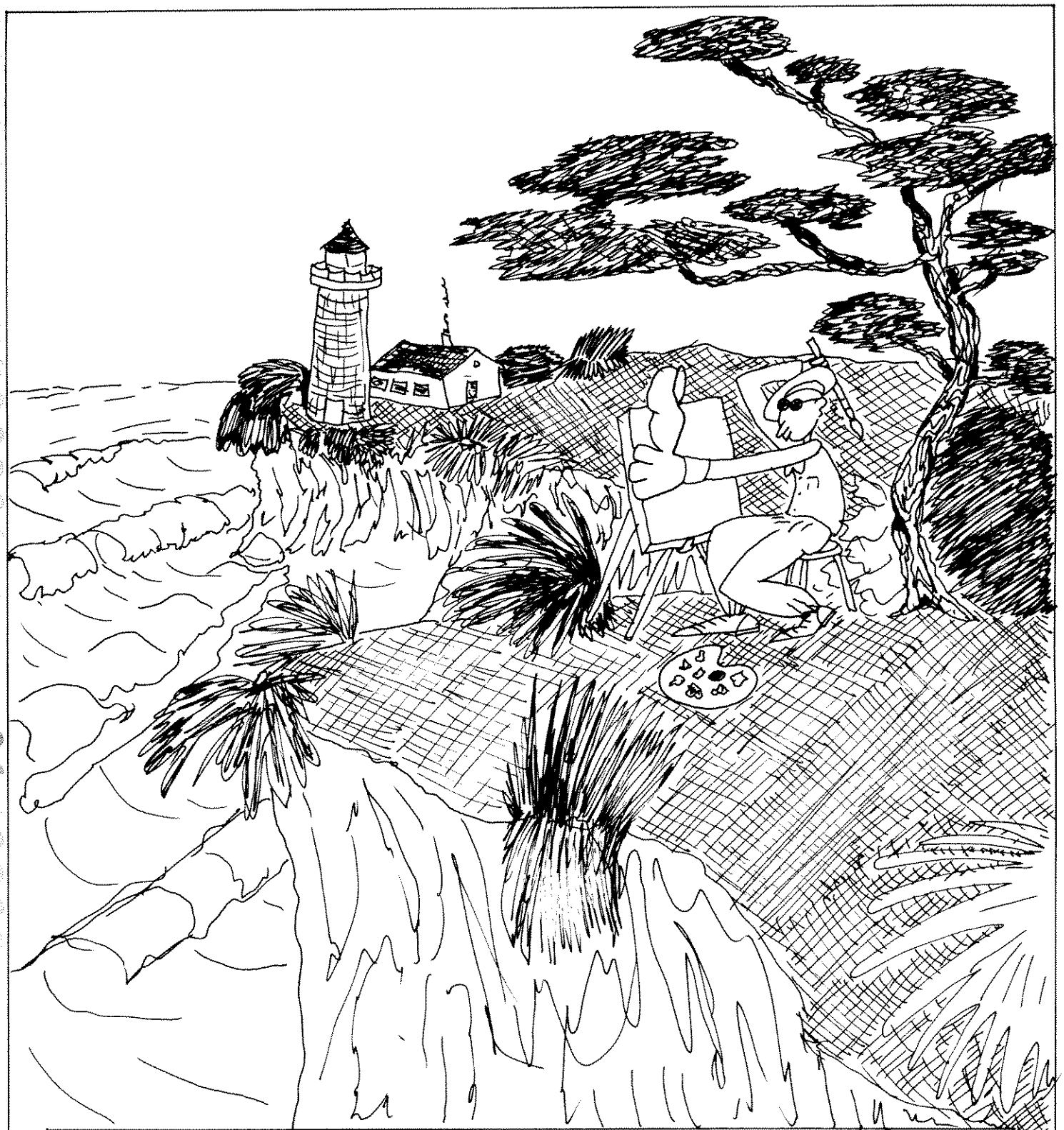
You Are Here...

...because you want to learn about your HP–11C or HP–15C.

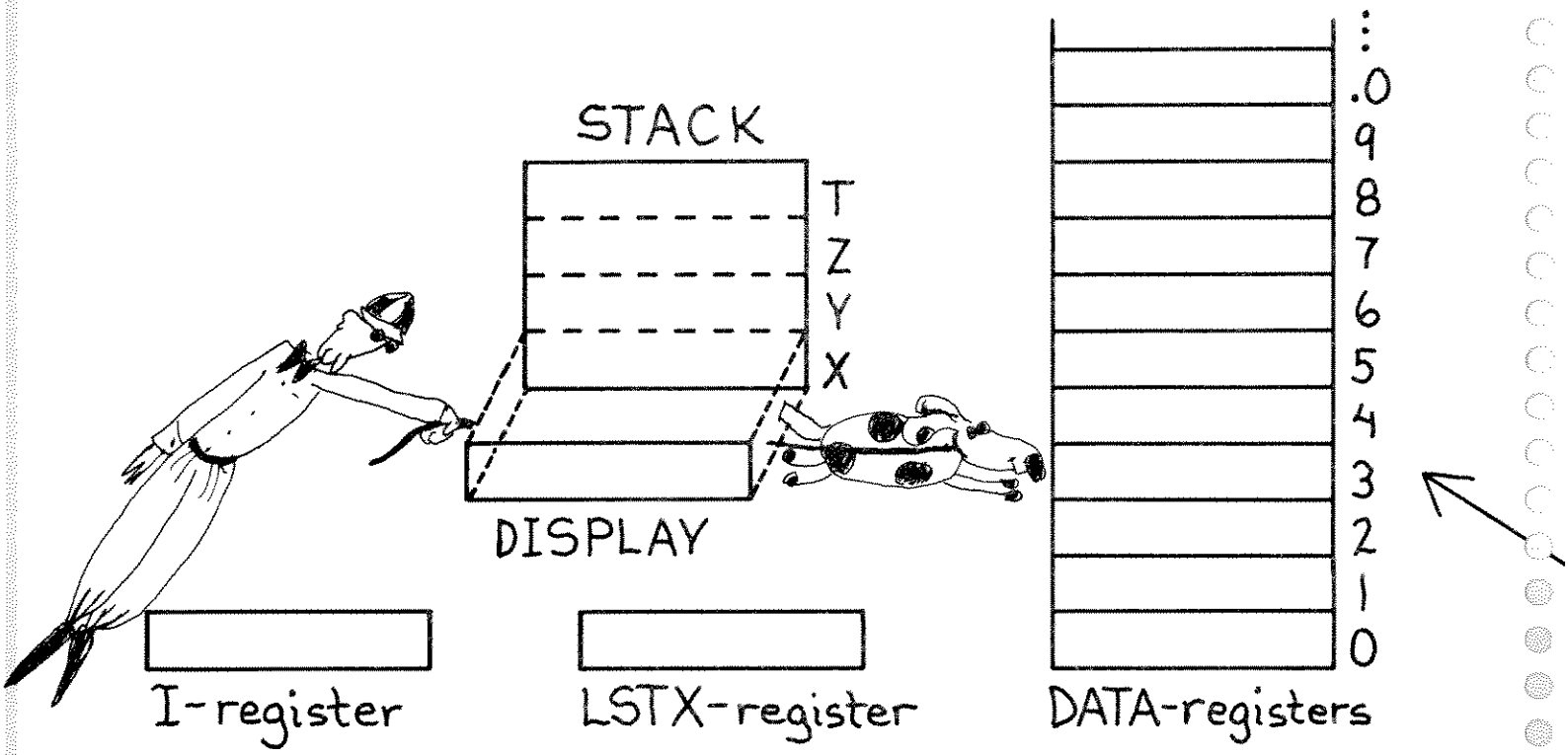
And you're in luck! You have found the best and quickest way to learn about your calculator. This course will take you on what may be the most incredible learning experience of your life(!)

This course is self–explanatory. Just follow the directions as you work through it. If you already know the material in some places, you'll be told to skip ahead. And it doesn't matter how long you take to learn each concept, so just sit back, relax, and learn at your own pace.

Well now.... The best place to start is probably at the very beginning of the story.... ——————>



How to Picture It



A PICTURE OF YOUR CALCULATOR'S MEMORY

As you may have guessed, your calculator has some memory of its own, where it can store numbers or program instructions. But how does that memory work? And what does it look like?

Actually, it's just a tangle of wires and circuits, but that's not a very useful way to visualize it. The above picture is a better way.

Each of those boxes represents a "register" in your calculator. A register is simply a place where a number can be stored for later use. Every register has a name, and you will be learning these register names and how to use them.

(If you already know all about this picture, and you don't need an introduction to each of these registers, then turn to page 12.)

DATA REGISTERS

Take a look at this block of registers. These registers are called data registers (or data storage registers). "Data" is simply another word for "numbers," so a data register is a place in your calculator where you can store a number (only one number at a time).

Data registers are "named" with numbers, starting at 0 and going up. You will use these "number names" to refer to the registers in the calculator. But notice that instead of using the names 10 through 19 where you would expect them, HP chose to use .0 through .9. That's OK; it's simply another way to number them. Just keep in mind that the register named .0 can sometimes be called register 10; register .1 can sometimes be called register 11; etc.

As you know, a data register can hold one number at a time. And when you store a number in a data register, you replace the number that was there before.

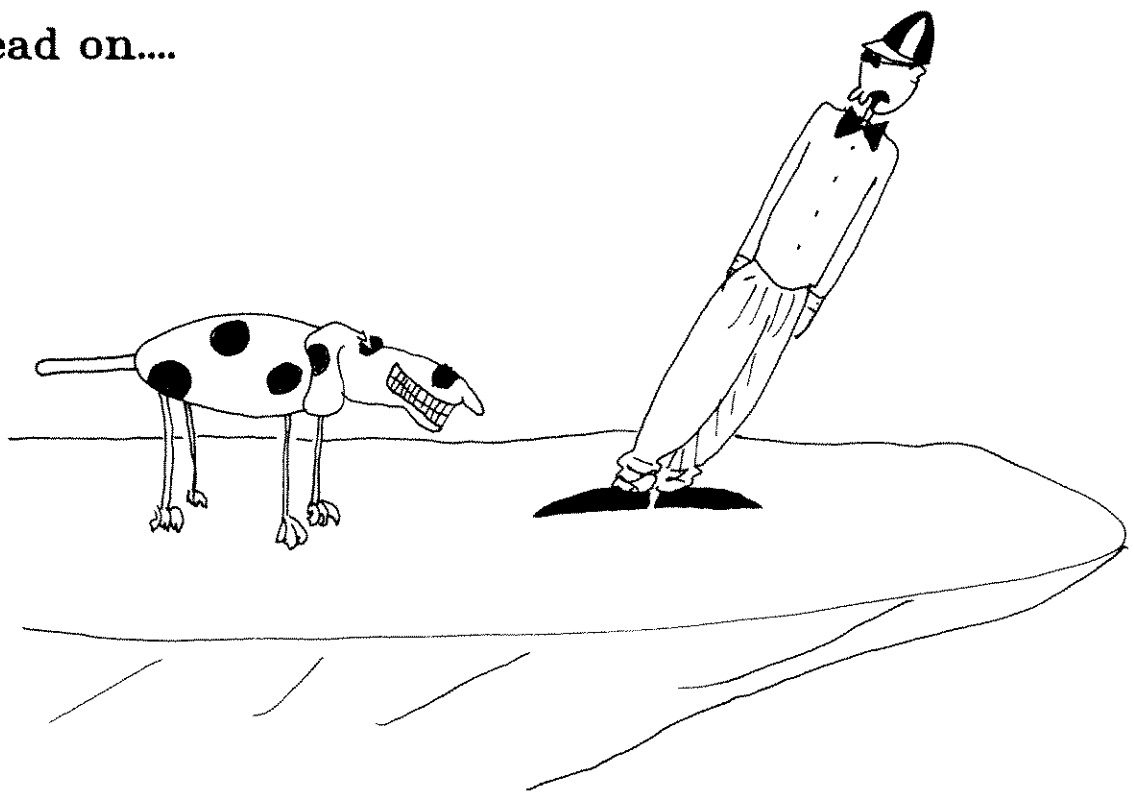
The HP–11C has a maximum of 20 data registers (0 to .9).

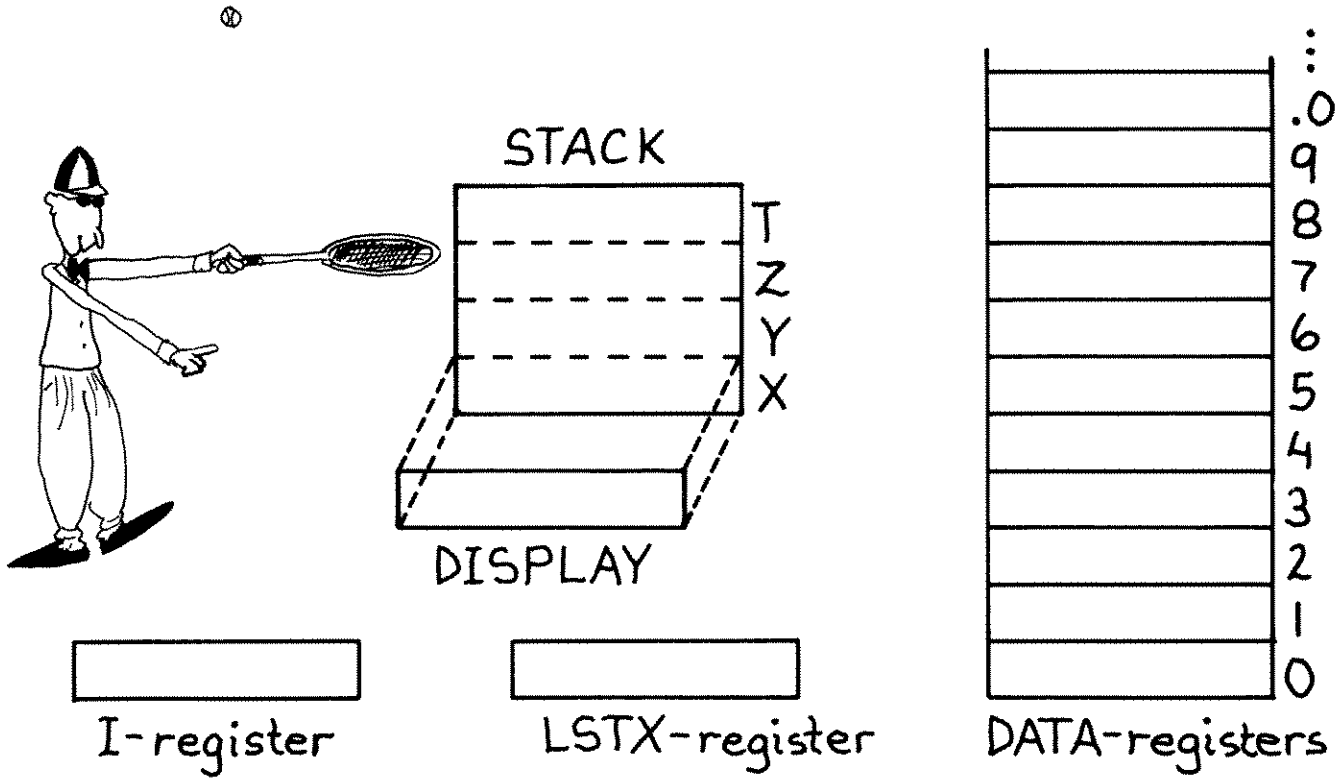
The HP–15C can have a maximum of 66 data registers (0 to .9 and 20 to 65).

But that says nothing about the number of data registers you have in your calculator right now. That number will vary. You can (sometimes unknowingly) change the amount of data registers. In fact, right now (even as you read this!), you may not have all of the 20 data registers 0 through .9 available for storing data.

Keep this in mind, but don't worry about the details right now. By the end of this course you'll know all about adjusting the amount of data registers.

So, read on....



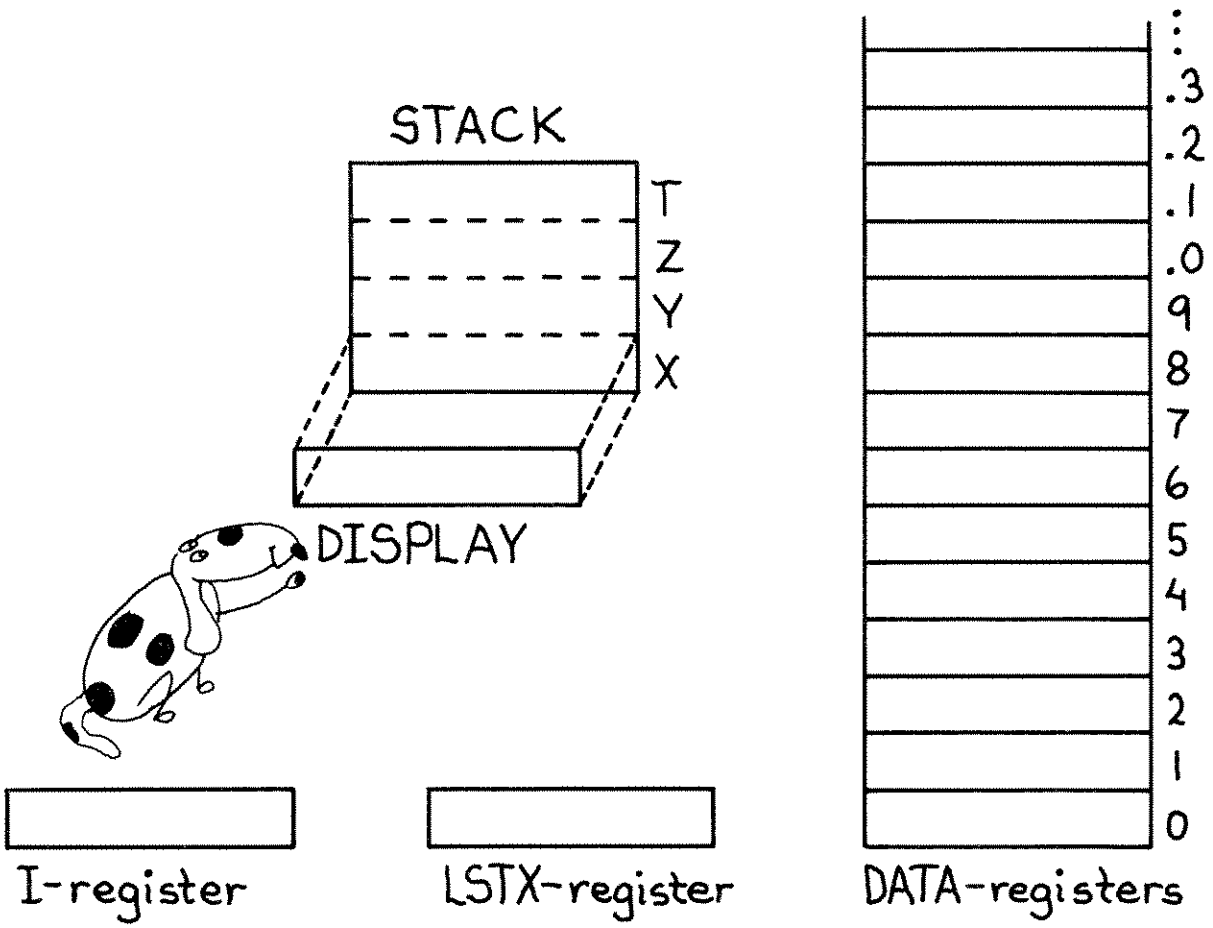


THE STACK

Next, take a look at the stack registers. The stack is simply a set of data registers that work together automatically when you are doing arithmetic and calculations.

Since these registers are just data registers, each of them can hold only one number each. But because they are linked together to form a "stack," their names are different from normal data registers: X, Y, Z, T, and LASTX.

Because they are used for arithmetic, these stack registers are the "workhorses"--the most frequently used registers--in your HP-11C or HP-15C.

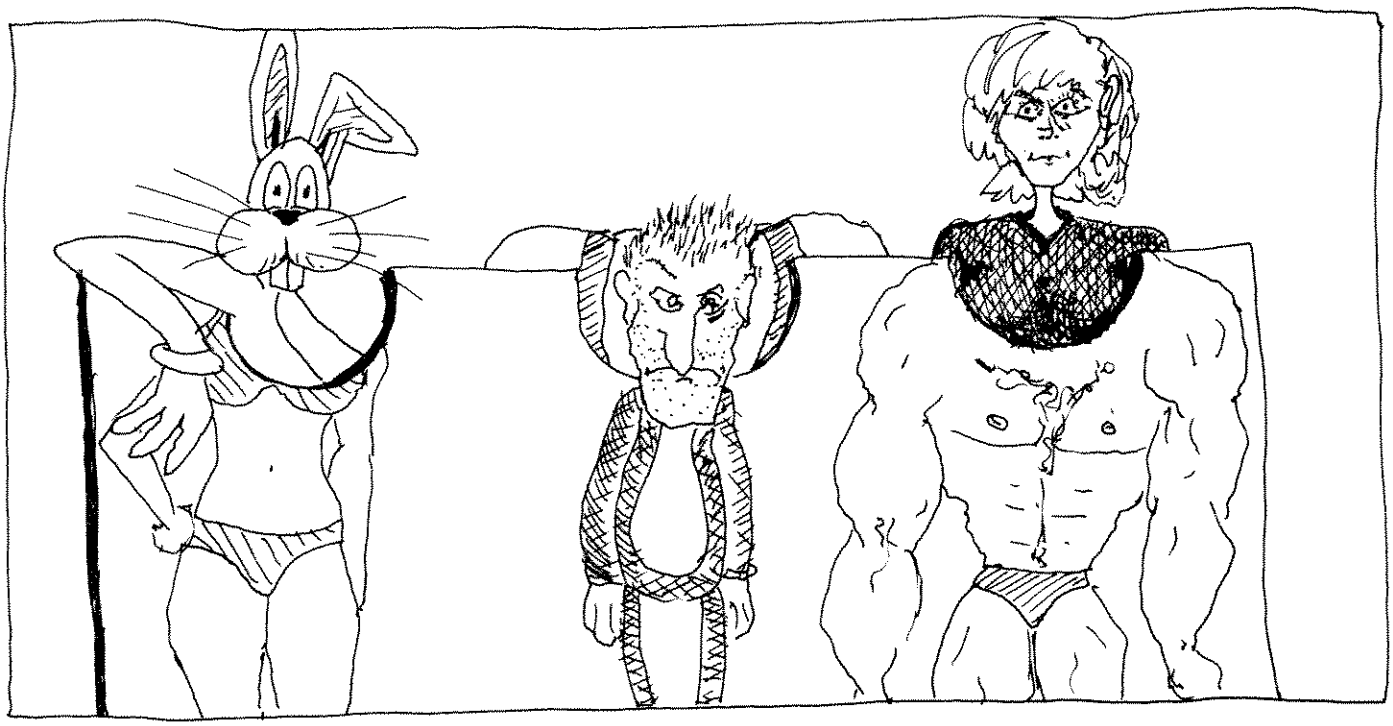


THE DISPLAY

Now notice the display. The display is the window of your calculator. It's what you look through to "see into" the machine. It's shown here positioned over the X–register. The display is ALWAYS positioned over the X–register (the bottom register of the stack).

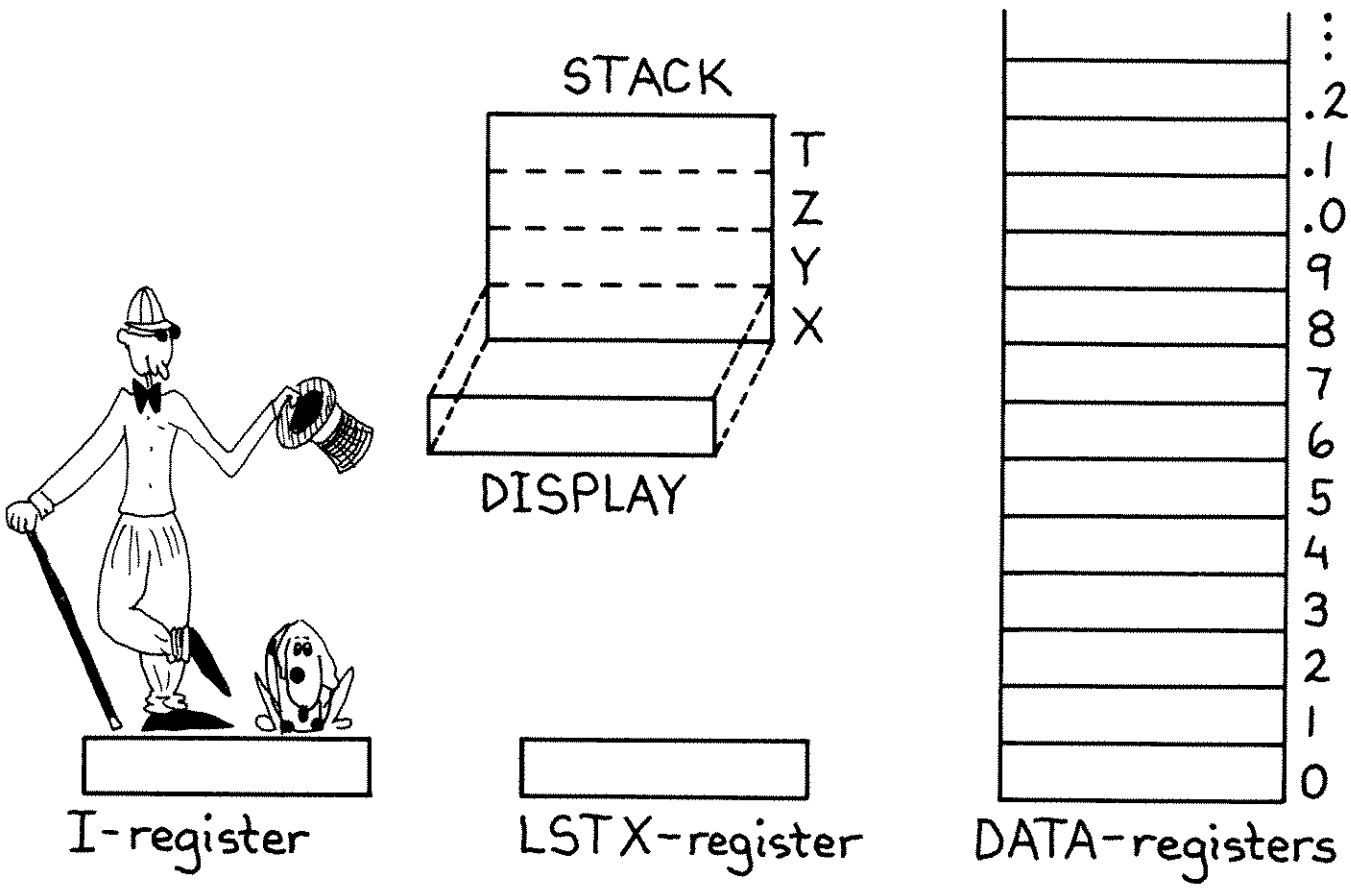
REMEMBER! WHEN YOU LOOK AT THE NUMBER IN THE DISPLAY, YOU ARE LOOKING AT THE NUMBER IN THE X–REGISTER, ALWAYS!

So why think of two separate things (the display and the X–register) if the display is always positioned over the X–register? The display is always showing the number in the X–register, so they're really the same thing, right?



Wrong!

The display is really a separate "window." Sometimes it's partly shut; that is, it doesn't always show you the entirety of the number in the X–register. Instead, it may show you a rounded version of that number. In fact, the display will show you only as many decimal places as you tell it to (and you'll soon learn how to adjust this). But remember, it's only the display that is doing any rounding. The number in the X–register is never rounded; it's always a complete, 10–digit number.



THE I–REGISTER

Finally, look at the picture of your calculator's I–register. It's really just another data register, but it's named with the letter "I," because, like the stack registers, the I–register has a special purpose. The number in the I–register can mean more to your calculator than "just another pretty number."

This special meaning of the I–register will be discussed in gory detail in a later section, so again, just keep it in mind for right now.

POP QUIZ

(Pop Quizzes appear periodically on pertinent pages to keep you primed!)

1. What's a data register?

2. How many data registers are named with numbers?

3. What kind of register is the T–register?
   The I–register?

4. What is the display?

POP ANSWERS

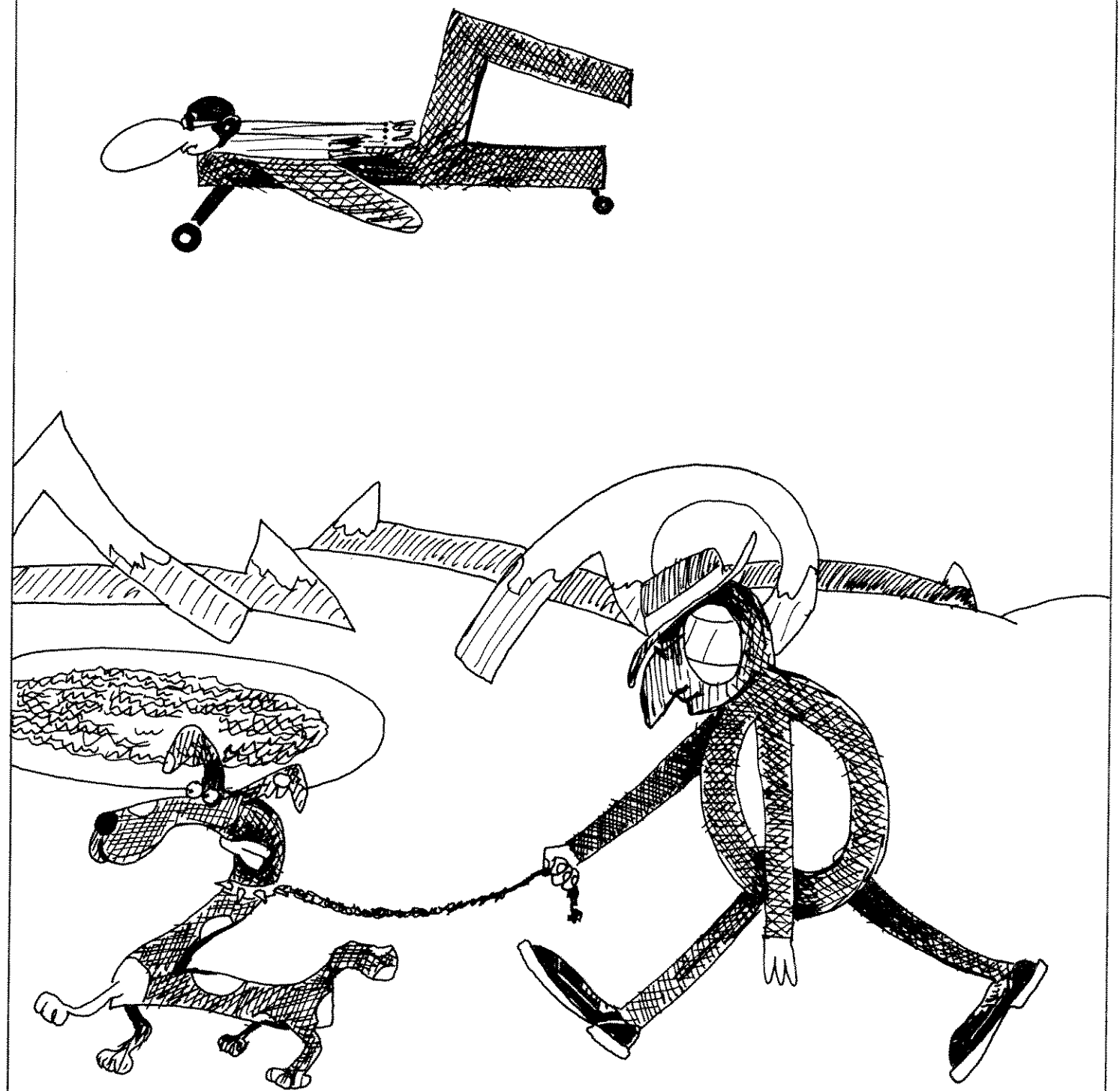
1. A data register is a place to hold data (numbers). One data register can hold one number at a time (page 6).

2. The amount of numbered data registers can vary, depending on how you have adjusted your calculator. The HP–11C has a maximum of 20; the HP–15C can have up to 66 (page 7).

3. The T–register is one of the stack–registers (the "Top" one). The I–register is a special register set aside for certain kinds of operations. Both of these are data registers (page 8).

4. The display is the window you look through to see the X–register. Sometimes this window may be partly "shut," thus rounding the number that you see, but the rounding happens ONLY in the display––not in the X–register (pages 9–10).

Did you make it through your first quiz without problems? It's important to have a good mental picture of the "insides" of your calculator, so if you need to go back to page 5 and give it another pass, do so now––before you go on.



Numbers and Functions

KEYING IN NUMBERS

Now that you know how to visualize the "insides" of your calculator, the next step is to put numbers into it.

First, turn it on: the ON key is nestled in the lower left hand corner of the keyboard (it's also the "off" key).

If you have an HP–15C and a little 'C' appears in your display, press the keys g CF 8 (the CF key is also the 5 key).

Now, to get a number into your calculator, you just "key it in." The number you key in will be stored in the X–register (and it will show in the display, right?).

You don't have to press any other keys.

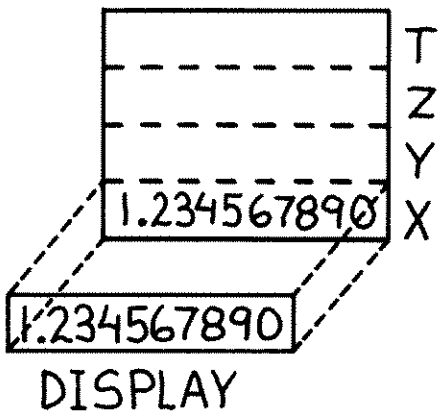
Try this: Store the number 1.234567890 in the X–register.

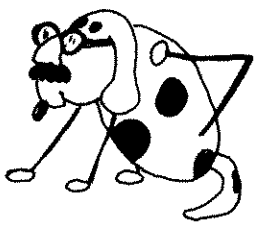
Solution: Press 1 . 2 3 4 5 6 7 8 9 0

That's all. You're done!

ADJUSTING THE NUMBER OF DECIMAL PLACES

Take another look at the display:



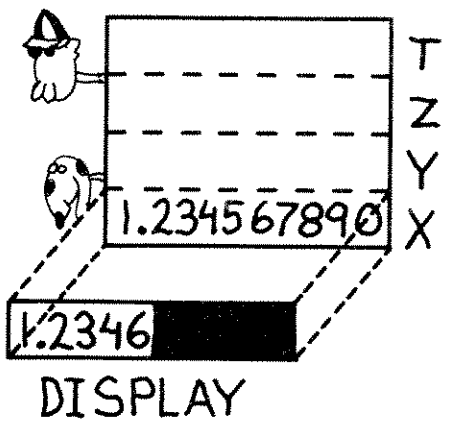


As you already know, the display is like a window over the X–register. And as you may have noticed, whenever you key in a number, this window slides open so you can see exactly what you're keying into the X–register.

But you usually don't need to see every decimal place of every calculated result; and sure enough, as soon as you perform any operation on that number, the window "closes" partly, to show you only a certain number of digits.

Try this: Set the display to show you only four decimal places of the number 1.234567890.

Solution: Press f FIX 4
(FIX is the gold printing on the 7 key)



The window "closes down" so that the number in the display becomes 1.2346, but the number in the X–register remains 1.234567890.

The last digit is rounded up to 6 in the display.

BUT REMEMBER! The display is doing this rounding. The number in the X–register is NOT rounded, and any calculation will use ALL 10 digits of this number. That's the way it is with HP calculators. They use and store 10–digit numbers––ALWAYS.

The important thing here is that you learn how to change the display setting any time you wish. So...

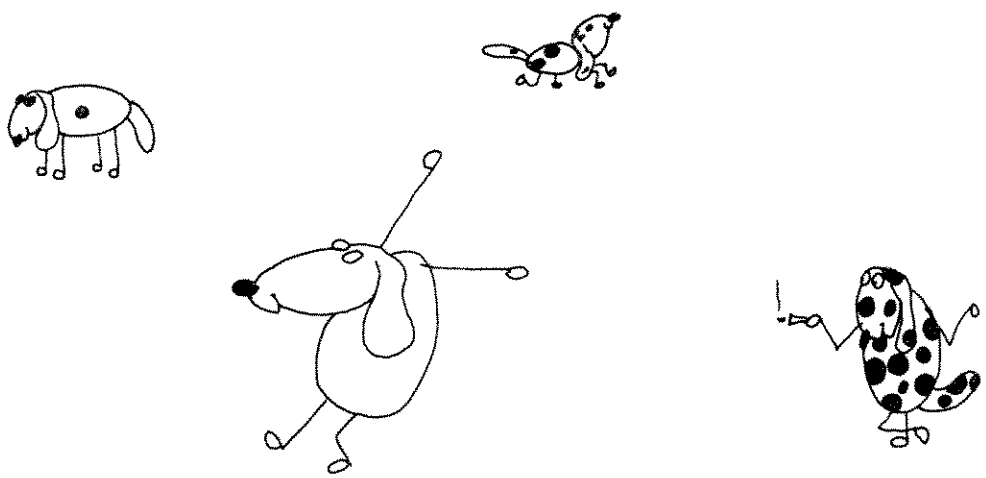
Try this: Adjust the display to show you 2 decimal places.

Solution: f FIX 2

Now this: Set the display to show 9 decimal places.

Solution: f FIX 9

Get the idea? Good. Now, set the display to whatever is comfortable for you; and remember, to properly compare your answers with ours, you should set your display to the same number of digits as you see in the printed answer, OK?



BEYOND THE X–REGISTER

Try this: Key in 100 to the X–register.

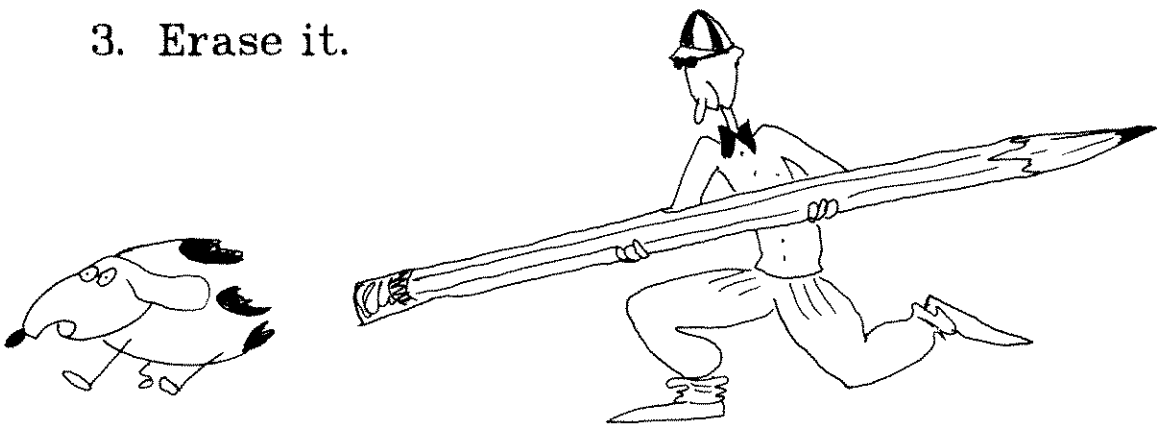
Solution: Press 1 0 0. That's all.

Once you have a number in the X–register, there are three things you can do with that number:

1. Store it in another register.

2. Use it to perform some calculation.

3. Erase it.



The first of these options is to store that number 100 somewhere else. (If you already know about the STO and RCL keys, turn to page 22.)

STORING NUMBERS

Try, for example: Put that 100 into register 2.

Solution: STO 2

Notice what this STO (STOre) key does: the number being stored is ALWAYS the one that appears in the X–register.

Also, the STO process is a copying process, not a transferring process. That 100 is now in register 2, but it's still in the X–register as well (you still see 100 in the display).

So to store into a numbered register, all you do is press STO and then name the register: STO 1, STO .3, etc., and as long as the register you name is actually there, the number will be stored.

The same is true for the I–register. If you want to store a copy of that 100 (which is still in the X–register) into the I–register, just press STO I. Try it.

RECALLING NUMBERS

Now that you know how to move numbers from the X–register to other registers, try moving numbers from those other registers back to the X–register. To do this, you use the RCL (ReCaLl) key.

This key recalls a number––from any register you name––back to the X–register.

Key a 5 into the X–register.

Try this: Recall that 100 from register 2 back to the X–register.

Solution: RCL 2

RCL makes copies of numbers––just like STO does. After you press RCL 2, there's a 100 in both register 2 and the X–register.

STO and RCL allow you to move numbers from one register to another––play "musical registers," one might say. For example, if you want to move a number from register 2 to register 0, you press RCL 2, STO 0.

Here's that list again. Once you get a number into the X–register (either by recalling it or keying it in), you can:

1. Store it in another register.

2. Use it in some calculation.

3. Erase it.

At this point, you know all about the first item in this list. The second item––performing calculations––is essentially the meat of this course. For an introduction, take a look at one of the simpler calculations you can perform on a number as it sits unsuspectingly in the X–register.

Try this: Change that 100 in the X–register to –100.

Solution: CHS (do NOT use the − key). CHS is the "CHange Sign" key, and it always operates on the number in the X–register.

Press CHS again. What happens?

This is just one example to show how most of the other calculation keys work. They usually operate on the number in the X–register (and sometimes other registers, too).

FUNCTIONS

A function is simply any operation that the calculator performs (except for social functions, which it does not perform). A myriad of functions --printed in gold, white, and blue--is available to you on your keyboard. Here are some examples:

Try this: Find $10^{3.77}$

Solution: 3.77 [10ˣ] (Answer: 5888.437)

The [10ˣ] function does just what its name implies--it raises 10 to the power of the value in the X-register. You key in 3.77, press [10ˣ], and the calculator raises 10 to the 3.77th power and places the answer in the X-register.

Try this: Find the natural logarithm (LN) of 5888.437.

Solution: g LN

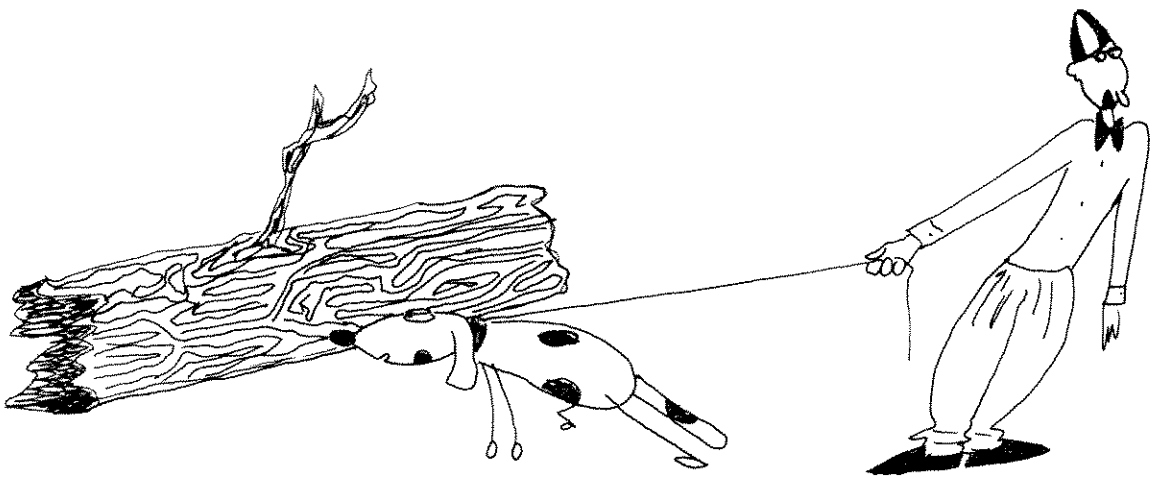
(If you got this solution with no problems, confused looks, or wrinkled forehead, you may want to skip the text on prefix keys and turn to page 27.)

PREFIX KEYS

There are two prefix keys on your calculator--the gold [f] key and the blue [g] key. Notice that almost every key has a white function printed on its face, a gold function printed above it, and a blue function printed below it.

If you want to use a gold function, you must first press the gold [f] key. Once you press the gold [f] key (and RELEASE it), all the keys take on their gold meanings. When you wanted to use the FIX function, you first pressed [f] to change the [7] key to the [FIX] key, right? The same is true for the [g] key. If you want to use a blue function, you must first press the blue [g] key.

Notice that when you press [g], a little g comes on in the display, indicating that all the keys have taken their blue meanings. Then, if you press [f], the g is replaced by an f in the display (and the keys then have their gold meanings).



On your calculator, the LN function always computes the natural log of the number in the X–register. So in the problem on page 24, since 5888.437 was already in the X–register (from the previous solution), all you had to do was find and use the LN function. You discovered that it was a blue function and thus required the blue [g] prefix key. By pressing [g] [LN], you obtained the LN (natural log) of 5888.437. The answer: 8.681 (if your display is set to FIX 3).

SOME OTHER GOOD THINGS TO KNOW

Try this: Key 10,000,000 into the X–register, using only two keystrokes.

Solution: EEX 7

The EEX key means Enter EXponent. Your display will show "1 07" when you finish the above solution. But that's the same number as 10,000,000, as you may already know. To prove this, just press ENTER or f FIX 2 to tell the calculator that you've finished keying in that number. It's 10,000,000 all right. (Notice that 10,000,000 has 7 zeros––no coincidence).

Expressing numbers in powers of 10 is called "scientific notation." You can set your display to use scientific notation with the SCI function. For example, press f SCI 2. The calculator display will now use scientific notation, with 2 decimal places. (Press f FIX 2 to get back to decimal notation.)

As another example, if you want to key in 50,000, you can either press 5 0 0 0 0 or you can press 5 EEX 4. By using the EEX function, you're expressing a number in powers of 10 (five times ten to the fourth power). On your calculator, scientific notation can save keystrokes and program lines, as you'll find out later.

Notice this also: As the 10,000,000 demonstrated, you don't have to press 1 before you press EEX. If you don't press a number before you press EEX, the calculator assumes you meant to press a 1.

There are some other convenient assumptions your calculator makes about the numbers you give it, and it's time to talk about some of them. (If you're not interested in trigonometry--SIN, COS, TAN, etc.--you may want to turn to page 32.)

Question: According to your calculator, what is the sine of 2?

Answer: Well, that depends. What do you mean by 2? Is that 2 degrees, 2 radians, or 2 grads? How do you tell the calculator what you mean?

The calculator has three trigonometric modes. Degrees mode is set by pressing g DEG; radians mode is set by pressing g RAD; and grads mode is set by pressing g GRD. The little words, "RAD" or "GRAD" will come up in the display when the calculator is set to each mode, respectively. Press each of these keys to see the effect it has on the display.

When the calculator has been set to RAD mode, it will assume that the number in the X–register is in radians. In DEG mode, it assumes degrees, and in GRAD mode, it assumes grads. There are (2 x PI) radians in a circle, or 360 degrees, or 400 grads. (The grads system seems like a logical one, but these authors have never seen it used.)

Now that you know about these three modes, ask that question again...

Question: What is the sine of 2 degrees?

Answer: (0.035)

Make sure your calculator is in degrees mode (If RAD or GRAD appears in your display, press g DEG). Then, to find the SINe of 2 degrees, press 2 SIN.

Try this: Find the sine of 2 radians.

Solution: g RAD 2 SIN (Answer: 0.909)

So remember, whenever you use any of the trigonometric functions, make sure you and your calculator are both in the same mode; if you mean degrees, be sure your calculator is set to degrees mode!

ANOTHER POP QUIZ

Again, be sure of the answers to these before you go on. (The answers are on the next page, as usual.)

1. **Which two keys allow you to move numbers from one register to another?**

2. **Which stack register is always involved with this storing and recalling?**

3. Do these functions "copy" or "transfer" numbers?

4. How do you change the sign of a number (from positive to negative or vice versa)? In which register must this number be in order to do this?

5. What is a prefix key, and how does it work?

6. How many prefix keys are on your calculator? Which ones are they?

MORE POP ANSWERS

1. The two keys are STO and RCL (pages 20–21).

2. These functions always use the X–register.

3. They copy numbers.

4. Use the CHS key. The number you want to change must be in the X–register (see page 22).

5. A prefix key is a key you must press (and release) before selecting the alternate (blue and gold) functions on any key (pages 25–26).

6. There are 2 prefix keys: the gold f and the blue g.

So, what do you know by now?

A. You know how to picture the insides of your calculator.

B. You know how to put numbers into it.

C. You know that once a number is in the X–register, you can:

    1. STOre it.
    2. Perform functions and calculations with it.
    3. Erase, or Clear it.

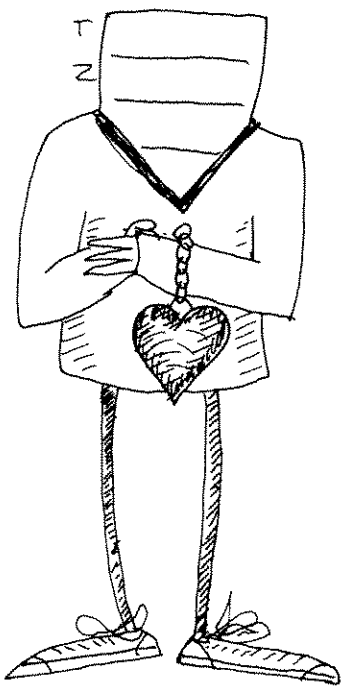
But you haven't really seen too many of those functions and calculations yet (nor how to clear numbers out of the calculator). You know how to set the display to a certain number of digits, and you know a few other things, but much more is yet to come.

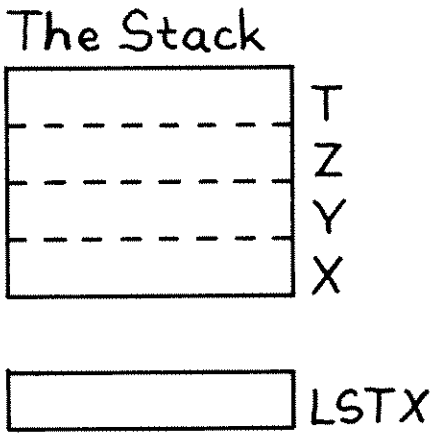
To know your HP calculator is to love it, and the first step toward REALLY knowing it is learning about the stack, that handy little hub of number–crunching that hums quietly beneath the display. This is where all that arithmetic is done.

Ah yes...

You've Got to Know Your Stack

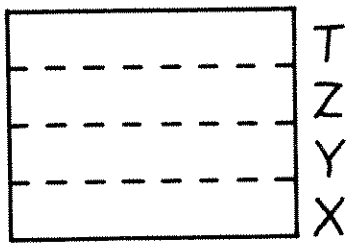
Whenever you do arithmetic on your HP calculator, you are using that set of five registers (named X, Y, Z, T, and LSTX) known as the stack. It's important that you understand how the stack works, because it is the heart of your HP machine.

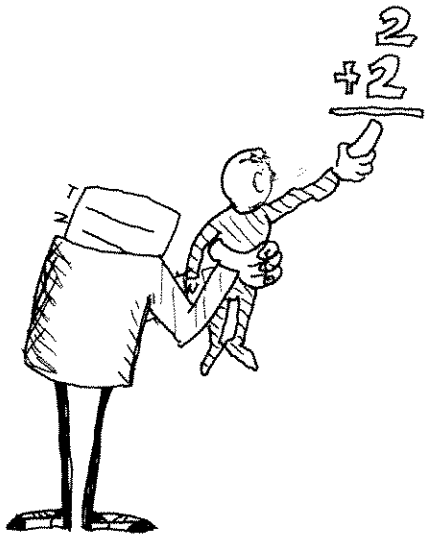




The LSTX–register is a rather unique member of the stack. It is important, but while you are learning the details of the X–, Y–, Z–, and T–registers, you can slip the LSTX–register into the far corners of your brain––to be retrieved later. So from here on, when you think about "the stack," just think about the X–, Y–, Z–, and T–registers. The LASTX–register is out of the picture––for now.

THE STACK





The stack is designed to help you solve arithmetic problems. It's that simple.

Try this: Solve the problem

$$\begin{array}{r} 24.27 \\ \times\ 3.86 \\ \hline \end{array}$$

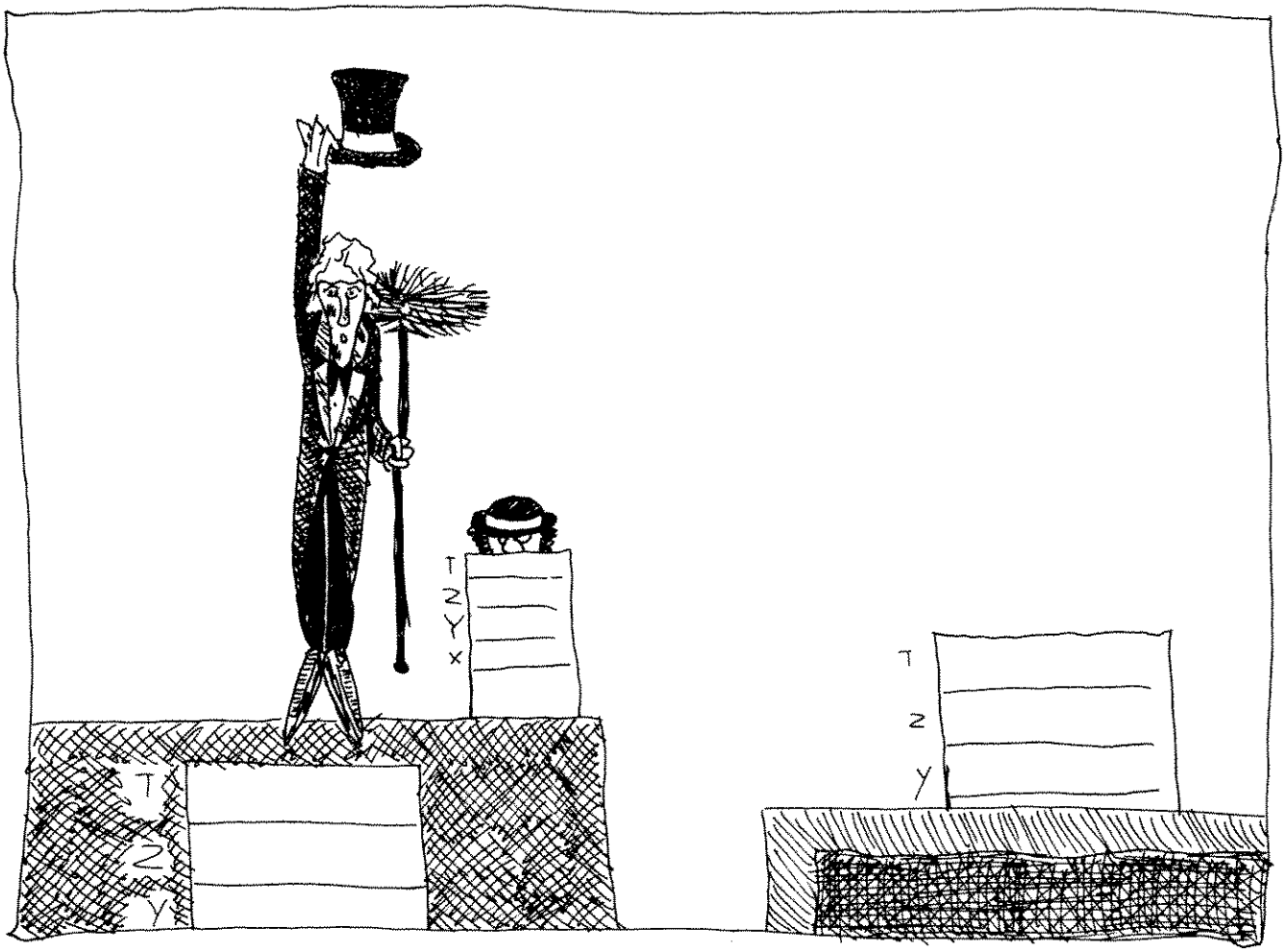
Solution: 24.27 ENTER 3.86 ⊠

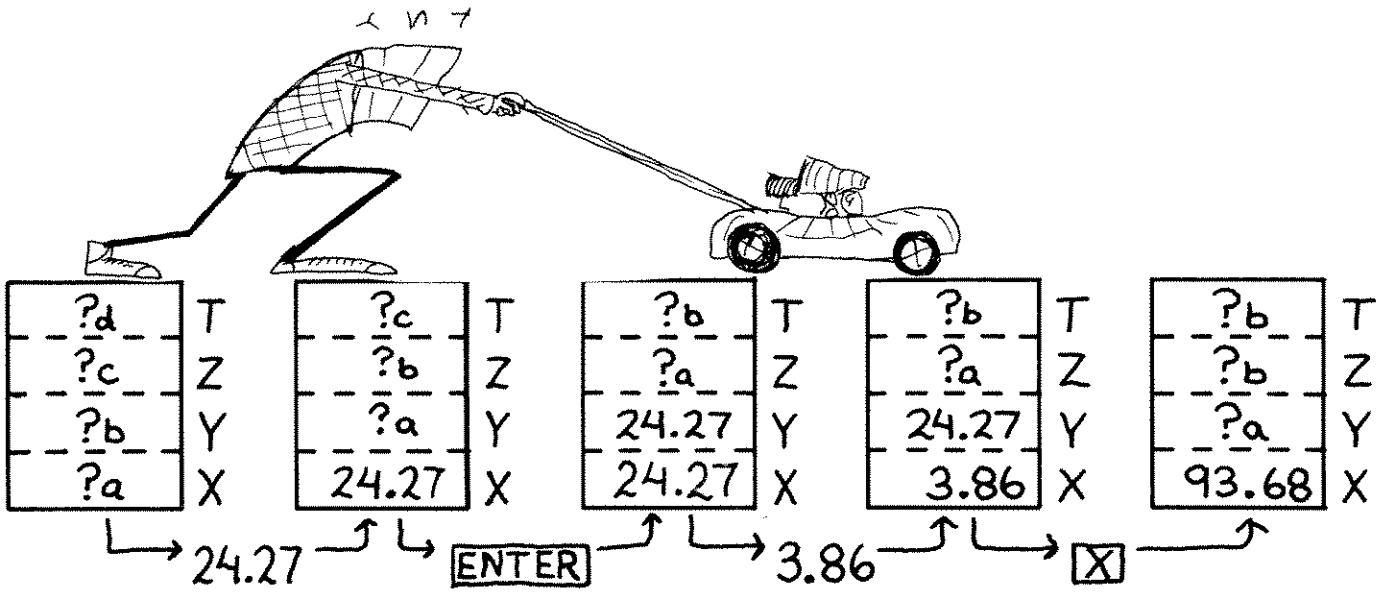
Answer: 93.68

You could probably come up with the above solution with little or no difficulty. When you first started playing with your HP calculator, you probably had the general idea that the ol' 2 ⊞ 2 ⊟ no longer applies (i.e. there's no ⊟ key on an HP calculator).

But do you know WHY the stack is the better way? And do you know what's happening in the stack? What happens when you press ENTER? Do you know what "stack–lift" is? Do you know when "stack–lift" is enabled? ...disabled?

If you answered yes to all of the above questions, turn to page 58. If you do so, however, you will miss an absolutely stunning explanation (complete with subtle, semi–hilarious illustrations) about all of the above. The choice is yours....





The above "stack–diagram" is a handy visual tool to demonstrate what's happening in the stack. The ?'s mean, "some number is there, but no one cares what it is." This is to remind you that there can be other numbers in the stack in addition to the ones you're working with, but they will never interfere. This means you don't have to clear all the stack registers everytime you start a new problem! And that saves both time and sore fingertips.

Now, there are four things to notice in the above diagram, and these four things represent the entire workings of the stack:

1. When you turn on your machine and key 24.27 into the X–register, the values in the stack are bumped up one notch. The value that was in the T–register (or Top–register) is gone. It has been bumped completely out of the stack and will never return (unless you key it back into the X–register, of course). When all the values in the stack are bumped up one notch like this, the process is called a "stack–lift."

2. When you press [ENTER], you cause another stack–lift. The values in the X–, Y–, and Z–registers are each bumped up one register. Notice, however, that the values in the X– and Y–registers are now the same; the X–register was copied into the Y–register.

3. Next, when you key in 3.86 AFTER pressing [ENTER], no stack–lift occurs. The 3.86 is simply written over the value that was previously in the X–register. Stack–lift is said to have been left "disabled" by the [ENTER] function.

4. Finally, when you press [X], the values in the X–, and Y–registers are multiplied, the answer is left in the X–register, and the other values (in the Z–, and T–registers) each drop one notch. Notice that the value in the T–register doesn't change.

There you have it! The workings of the stack in four easy steps. So, are you ready to write your first program?

Well, not really. First, you'd better get a little more practice with the stack. (Of course, if you're quite comfortable with the stack and ALL of its features, try page 58.)

ENTER

On page 40, Step 3 points out that ENTER leaves stack–lift disabled; that is, when you key in the 3.86, it replaces the number that was in the X–register, instead of "bumping" it up.

Take a closer look at these phrases:

1. Stack–lift is enabled.
2. Stack–lift is disabled.

The calculator's normal state is with stack–lift ENabled. That is, most of the time, when you key in a number (or recall one from a register), a stack–lift occurs.

But after the ENTER function, stack–lift is left DISabled. When you key in a number after ENTER, the stack doesn't lift.

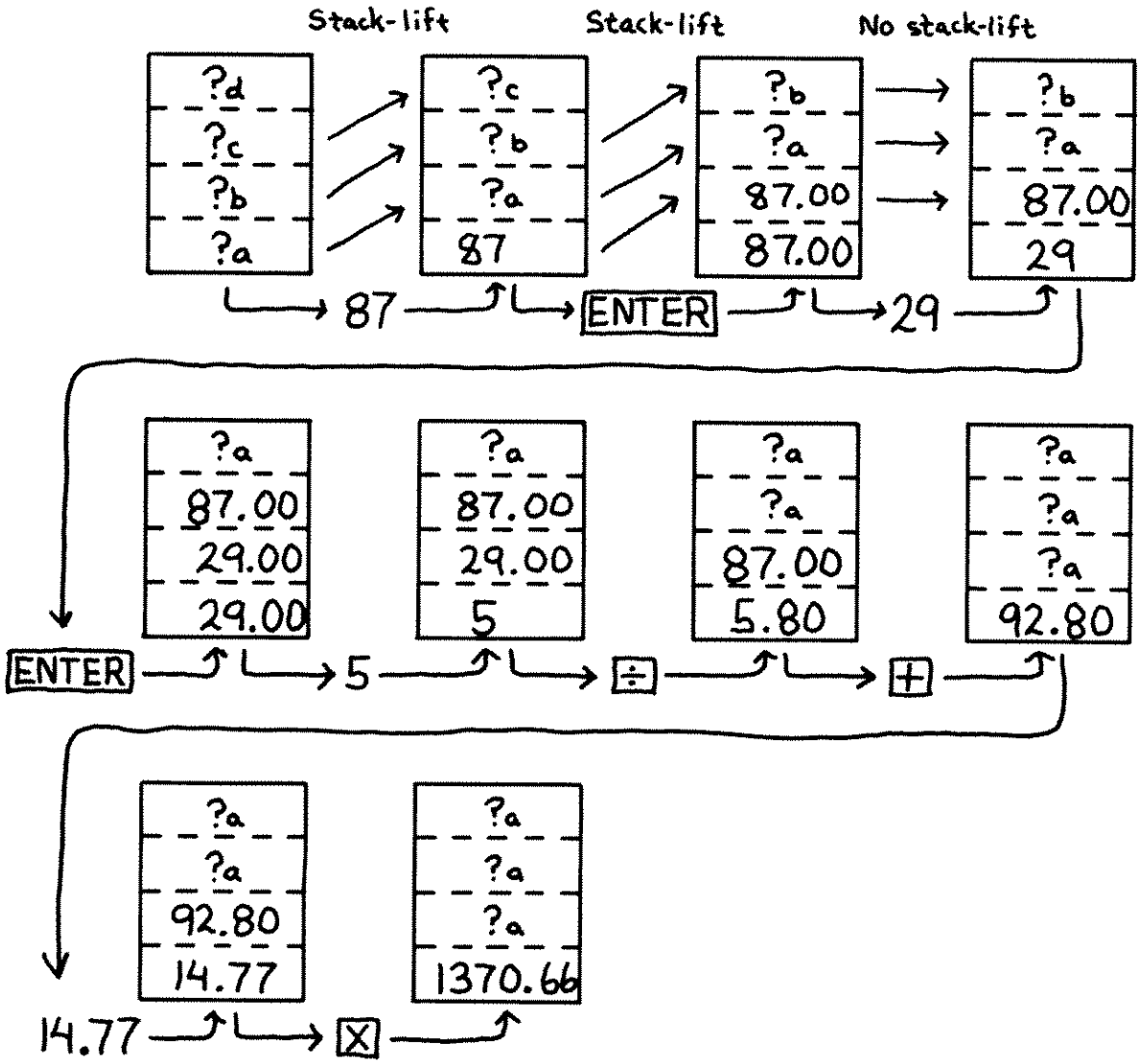
Armed with this knowledge, try another problem.

Try this: (87 + (29/5)) X 14.77

Solution: 87 [ENTER] 29 [ENTER] 5 [÷] [+] 14.77 [X]

Answer: 1370.66

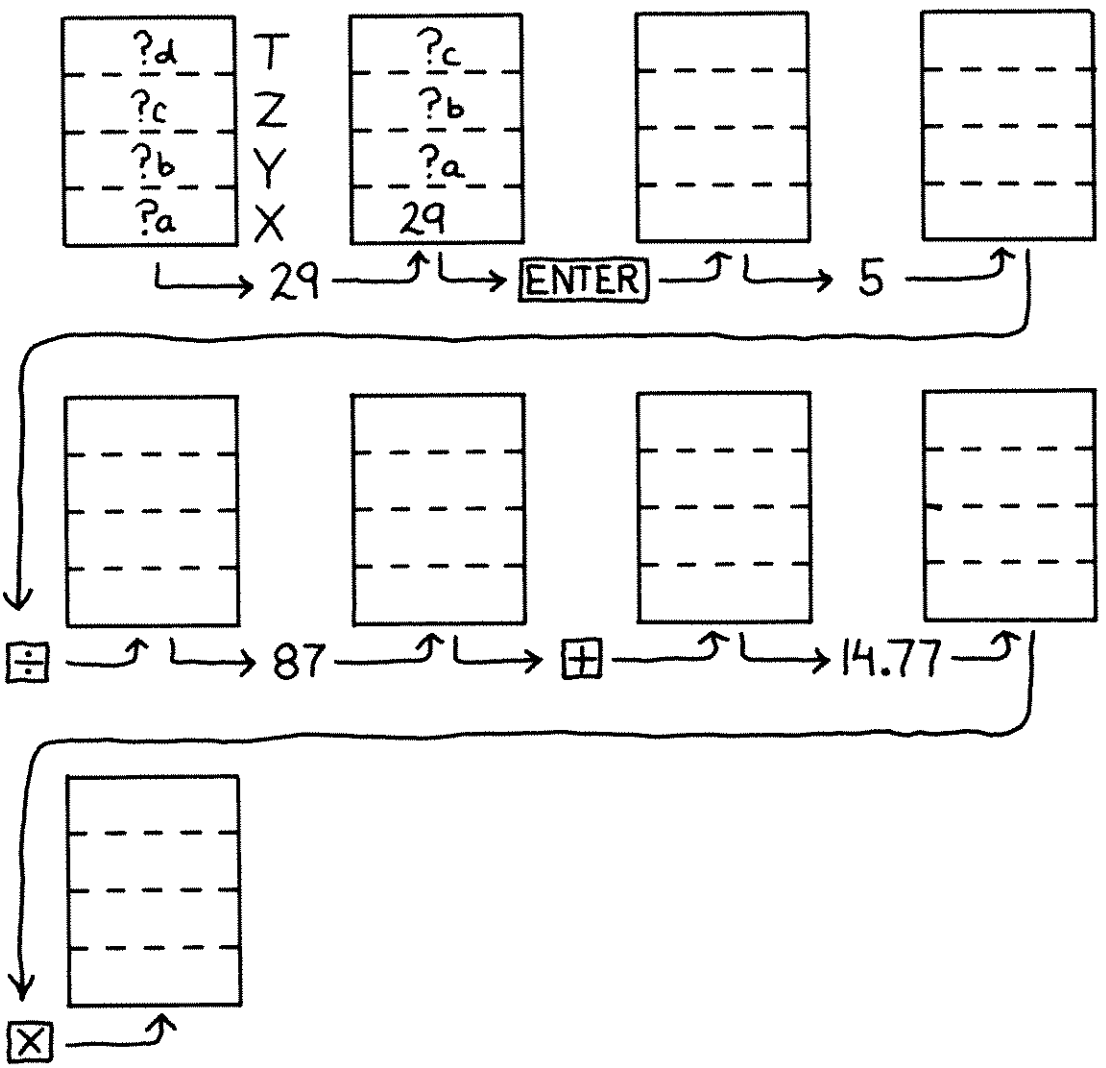
Look at this solution in the stack. Notice how the 87 stays up in the stack until it is used.



Here's another way to solve that problem:

29 ENTER 5 ÷ 87 + 14.77 X

Try to fill in the stack diagrams for the above solution.



REMEMBER! ENTER does two things (in this order):

1. It causes a stack–lift.
2. It disables stack–lift.

CLX

If you make a mistake while keying in a number, and that mistake is now sitting in the X–register (i.e. you see it in the display), you can replace that incorrect number in the X–register with the correct number (without disturbing the rest of the stack) by using the function CLX (Clear X).

Suppose, for example, you're working out the problem:

$$\frac{4.7 - \sqrt{(7.55)^2 - 4(2.9)(1.63)}}{2(2.9)}$$

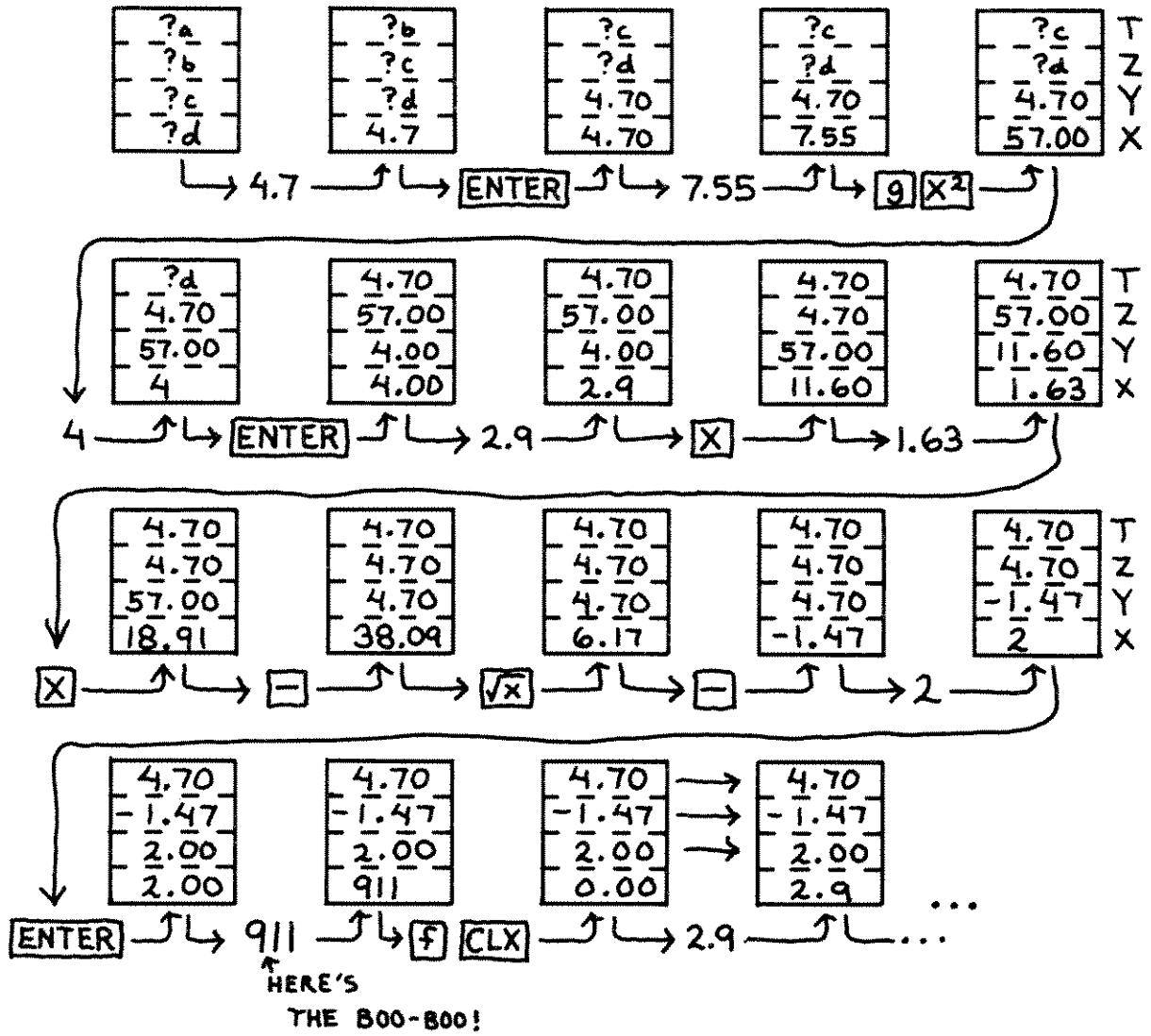
One set of keystrokes that will work for this problem is:

4.7 ENTER 7.55 g X^2 4 ENTER 2.9 X 1.63
X − $\sqrt{x}$ − 2 ENTER 2.9 X ÷

But suppose that while you are going through these keystrokes, your friend asks you what telephone number to dial in case of an emergency, because the backyard is on fire. So, right before you press that last X, you realize that you have keyed in 911 and not 2.9. How can you correct this error?

Answer: Press CLX.

Here's what the stack looks like as you go through the keystrokes:



The CLX function (CLear X) does two things (in this order):

1. It simply replaces the number in the X–register with a zero;

2. It DISABLES stack–lift, so that the next number you key in will simply replace the zero in the X–register, without disturbing the rest of the stack.

THE ⌫ (BACK-ARROW) FUNCTION

The ⌫ (back-arrow) function is another function you can use to correct errors. This key will remove one digit at a time from a number if it is currently being keyed into the display.

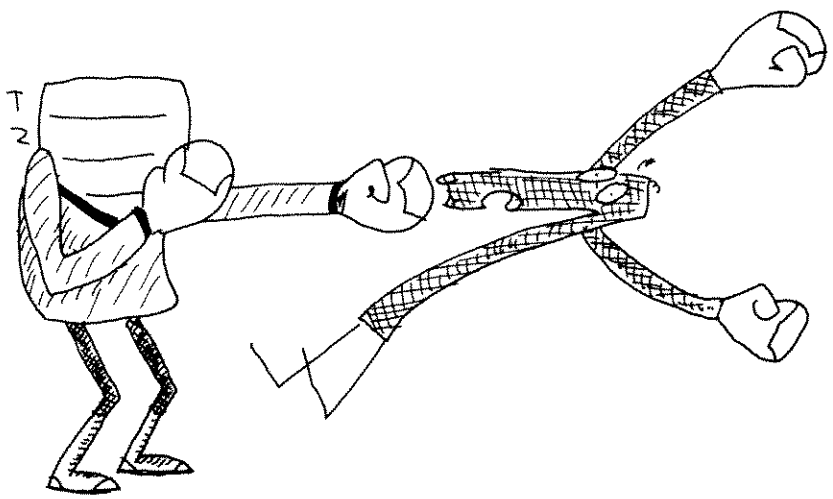
Take one common example: Suppose you are casually keying in some number while remembering the piano concerto you performed last week with the New York Philharmonic, and you hit a few wrong digits. You can correct this simply by stopping. Then, before you press any other keys, press the ⌫ key, and you will see the number in the display being reduced one digit at a time with each keystroke.

When there is only one digit of your "partly-built" number left in the display, or if you're not currently keying in any number, then pressing the ⌫ key is exactly like pressing g CLX. You'll get a zero in the X-register, and you'll DISable stack-lift.

Don't worry--you'll get comfortable with these details after some practice.

IN SUMMARY:

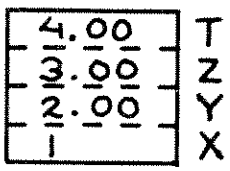
The main thing to remember about the stack is that ENTER and CLX are the only two commonly used stack operations that leave the stack DISabled. Most other operations will leave the stack ENabled. This may seem complicated, but it's really quite simple, once you have practiced it a little bit. It's like driving a car: you'll soon find that you're not consciously remembering what the stack is doing (bumping numbers to and fro), but that you know how it works nevertheless!



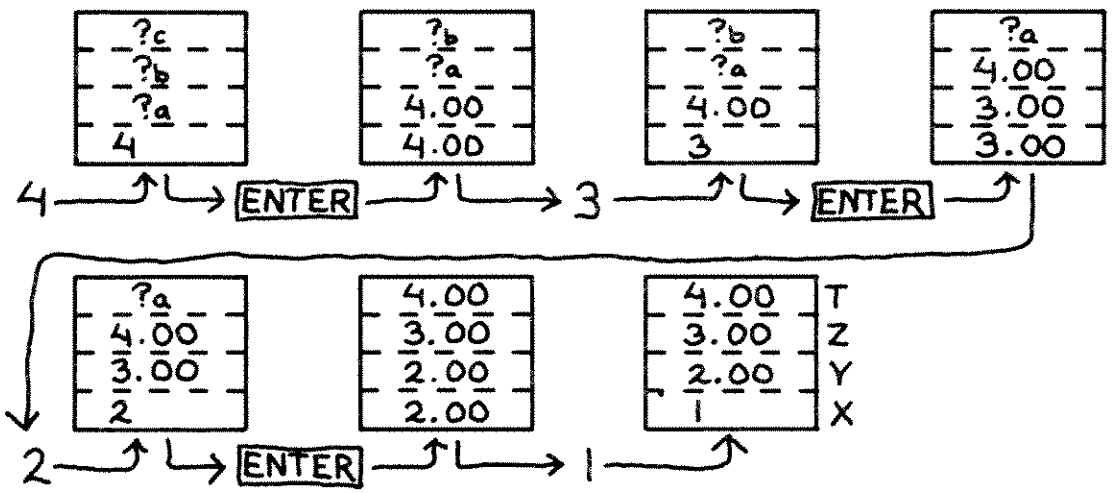
OTHER STACK OPERATIONS

There are three other functions that you can use to move numbers within the stack. These functions are X<>Y, R↓, and R↑. Simply by knowing their names, can you tell what these functions do?

Try this: Set up your stack like this:



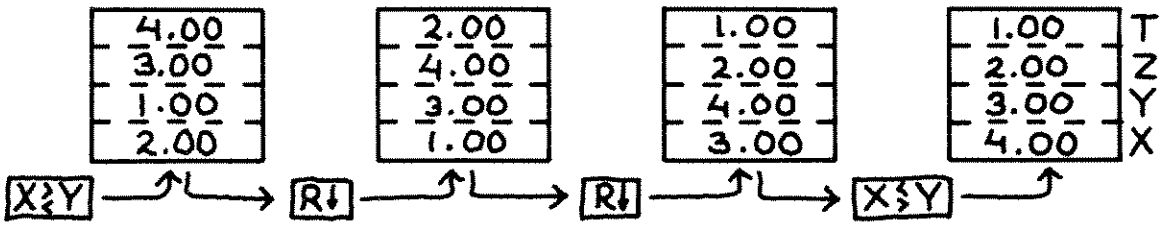
Solution: 4 ENTER 3 ENTER 2 ENTER 1



Now try this: **Without keying in any numbers, reverse the order of these values in the stack.**

Solution: X<>Y R↓ R↓ X<>Y

Here's what the stack does as you work through that solution:



As you can see, the R↓ (Roll Down) function rolls all the values in the stack down one register, and the value in the X–register is sent up to the T–register. The other function, R↑ (Roll Up), does just the opposite. You can see where these names "Roll Down" and "Roll Up" come from.

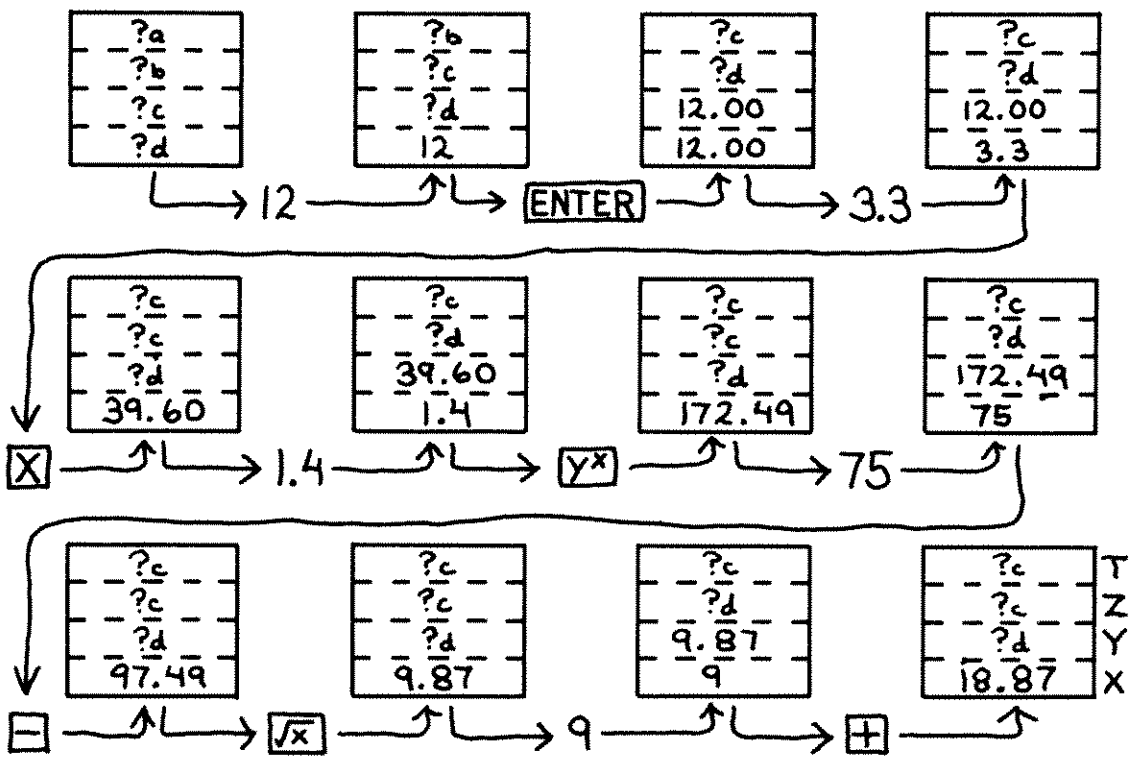
The function X<>Y (X exchange Y) does just what its name implies, too. It exchanges the contents of the X– and Y–registers.

MORE STACK PROBLEMS

Now, to become totally comfortable with the stack and its functions, you have to practice a little and look at a few stack diagrams. So relax, take your time, and you'll find that solving problems is no problem at all!

Try this: $9 + \sqrt{(12 \times 3.3)^{1.4} - 75}$

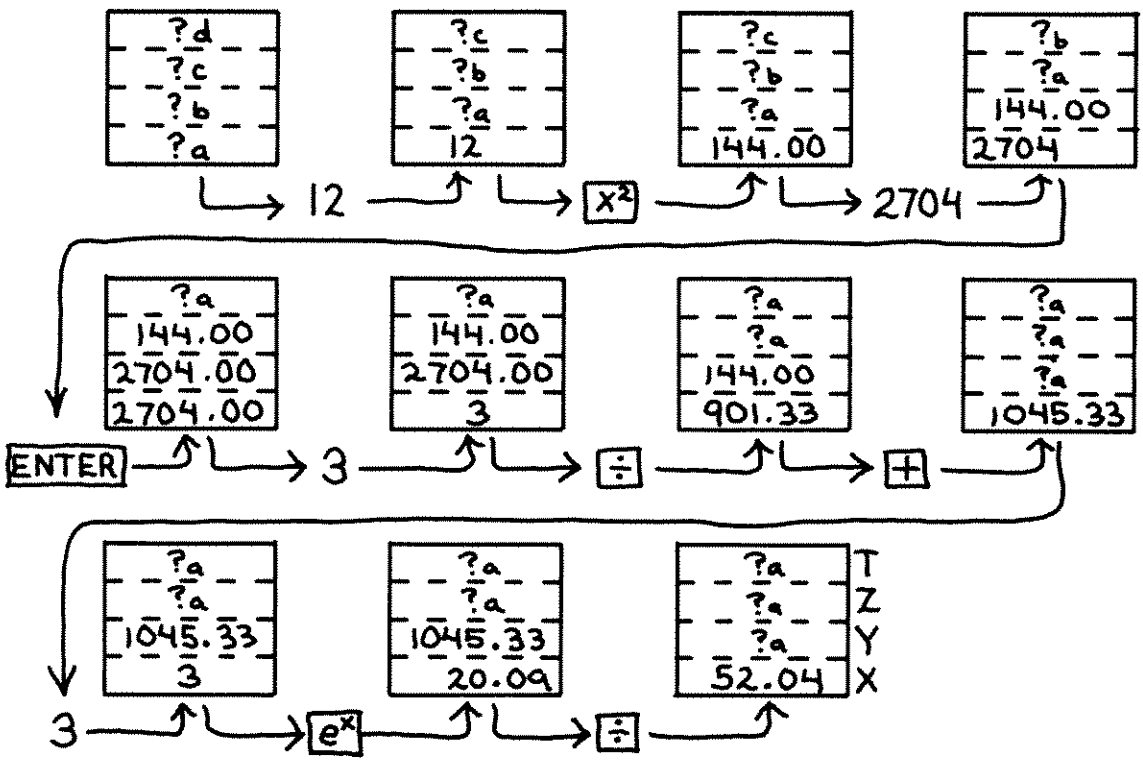
Solution: 18.87



Try this:

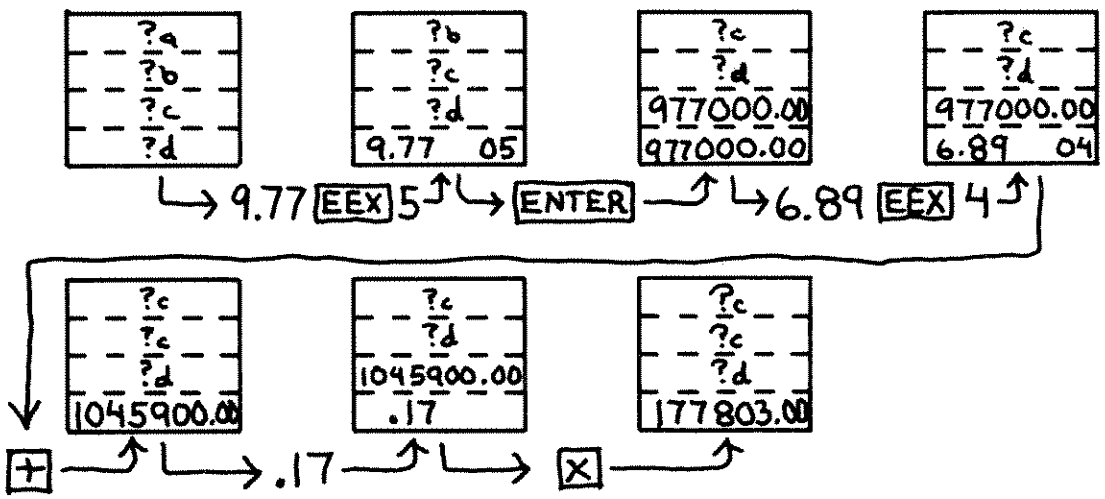
$$\frac{(12^2 + 2704/3)}{e^3}$$

Solution: 52.04



How about this? $0.17 \times ((9.770 \times 10^5) + (6.89 \times 10^4))$

Solution: 177,803.00



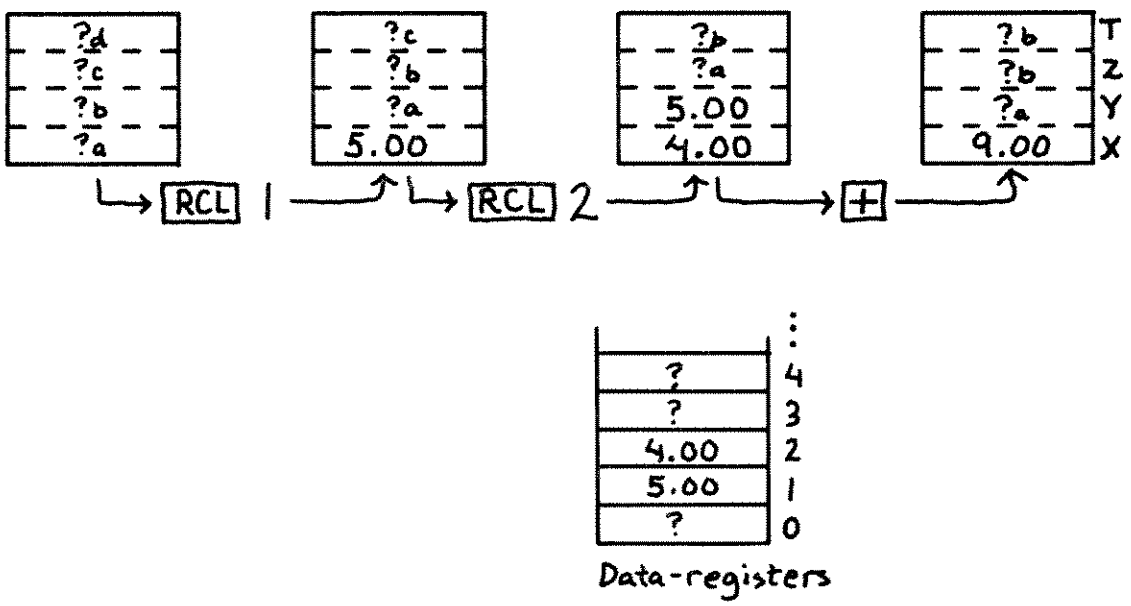
DATA REGISTERS AND THE STACK

Now take a look at how you can use numbers stored in data registers for arithmetic problems. For example, store a 5 in register 1 and a 4 in register 2 (5 STO 1; 4 STO 2). As you know, the STO and RCL functions will leave stack-lift enabled.

Try this: Add the number in register 1 to the number in register 2 and keep the result in the X-register.

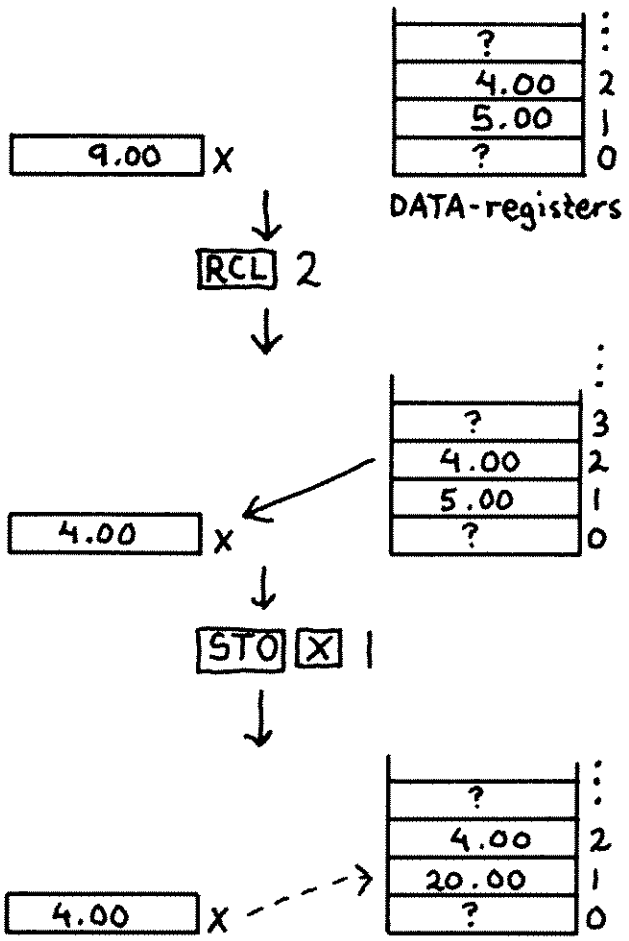
Solution: RCL 1 RCL 2 +

The stack looks like this:



Now this: Multiply the number in register 1 by the number in register 2, but this time let the result end up only in register 1. In other words, use "register arithmetic."

Solution: RCL 2 STO X 1



"Register arithmetic" is convenient for performing arithmetic and storing results at the same time. It really comes in handy when you're writing efficient programs, as you'll soon find out.

THE LSTX REGISTER

Finally, here is a quick look at that mysterious LSTX register.

The LSTX register is really just another data register. But what sets it apart is that the number in this register is always changing (automatically) as you do arithmetic on your calculator.

Whenever the number in the X–register is altered in any arithmetic operation (like + or ÷), the last X value is stored in the LSTX register. You can recall this value by pressing g LSTX.

In general, functions that simply move numbers around (like X<>Y and STO) don't save the last X value. But most functions that alter the number in the X–register will save the unaltered version in LSTX.

LSTX is a register you may not use much, but once in a while you may find it handy for correcting errors you make in your stack arithmetic.

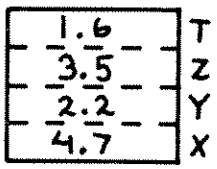
To recall to the X–register the value currently saved in LSTX, you would simply press g LSTX. It's that simple.

Notes

QUIZ

1. To double the number in the X–register, you can use the keystrokes [ENTER] 2 [X] (three keystrokes). Can you accomplish the same thing with only two keystrokes?

2. (Part 1) Without using the [ENTER] key, configure the stack as such:



then, (Part 2) without keying in any numbers, compute:

$$\frac{(3.5 - 2.2)^2 + 4.7}{1.6}$$

3. Which set of keystrokes will clear the stack (store zeros in all the stack registers X, Y, Z, and T)?

   a. [g] [CLX] [g] [CLX] [g] [CLX] [g] [CLX]

   b. [g] [CLX] 0 [g] [CLX] 0 [g] [CLX] 0 [g] [CLX] 0

   c. [CLX] [ENTER] [ENTER] [ENTER]

   d. 0 [R↓] [X] [X] [X]

4. True or False? [RCL] 1 [ENTER] [RCL] 2 [+]
   gives exactly the same result and leaves the stack set up exactly the same as [RCL] 1 [RCL] 2 [+].

ANSWERS

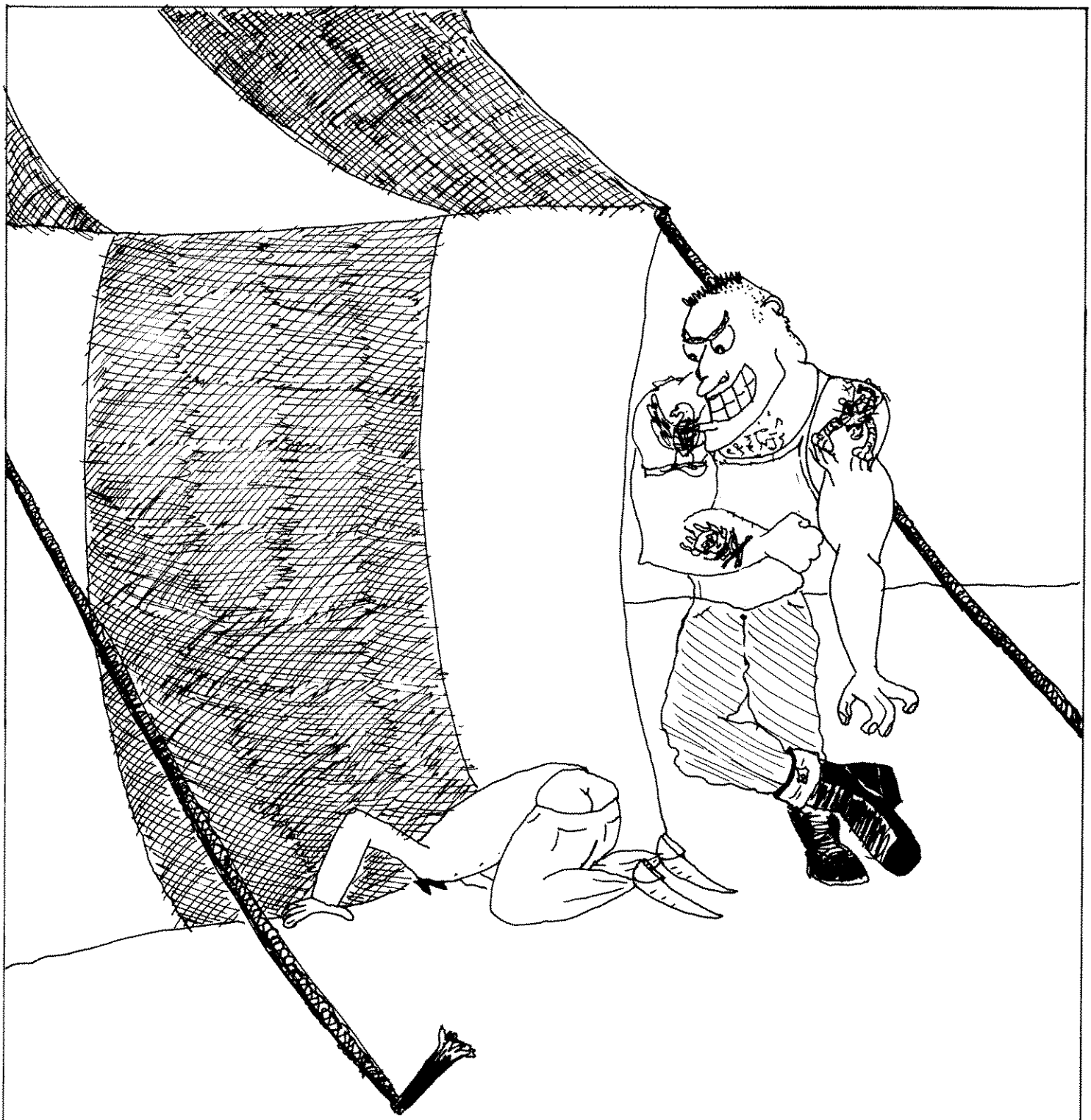
1. Yes, [ENTER] [+].

2. To set the stack up in part 1 (without pressing [ENTER]), 1.6 [X<>Y] [X<>Y] 3.5 [g] [RAD] 2.2 [g] [DEG] 4.7 is one possibility. The trick here is that the execution of almost any function leaves stack–lift enabled. Thus, when you do something like [g] [RAD] or [R↓] [R↑], the calculator assumes you've finished keying in the previous number, and stack–lift is enabled.

   For the second part, you can use the keystrokes [R↓] [−] [g] [X²] [g] [R↑] [+] [X<>Y] [÷] = 3.9938.

3. Both c and d are correct. Selections a and b very effectively clear the X–register. There's a rare chance that selection d can cause an error if the numbers in the stack are huge.

4. True. Pressing [ENTER] would be a waste of time and extreme effort in this case. Draw the stack diagrams if this isn't clear to you.



The Naked Program

Imagine that you've just finished a physics experiment and you have a list of fifty temperatures. But because your lab is ill–equipped, you had only Fahrenheit thermometers to use, so you measured all of those temperatures in degrees Fahrenheit. Now you need to convert them to more acceptable units––degrees Celsius.

The formula to do this conversion is:

$$\frac{5 \text{ X } ((\text{degrees F}) - 32)}{9}$$

$$= (\text{degrees C})$$

Try this: Figure out the keystrokes you need to convert 97 degrees Fahrenheit to degrees Celsius.

Solution: 97 ENTER 32 − 5 X 9 ÷

And now do the same conversion for 85 degrees Fahrenheit.

Solution: 85 ENTER 32 − 5 X 9 ÷

Well, running through these keystrokes fifty times (once for every temperature you took in your experiment) is going to be no fun at all. You're going to be repeating the same seven steps over and over for each temperature, and this will probably become a real drag.

But wouldn't it be nice if you could write a program so that all you had to key in was the Fahrenheit temperature, and then by pressing ONE other key, you could get the answer in Celsius? That is, wouldn't it be nice if you could write a program to solve this equation?

$$\frac{5 \times ((\text{INPUT}) - 32)}{9} = (\text{OUTPUT})$$

Good news....

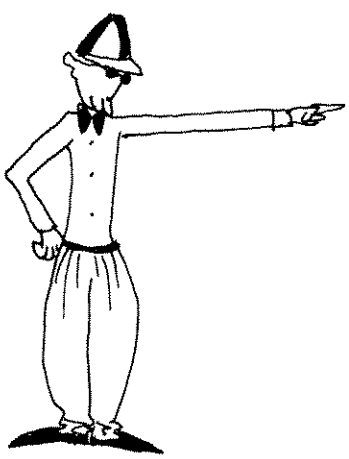
You have already written that program!

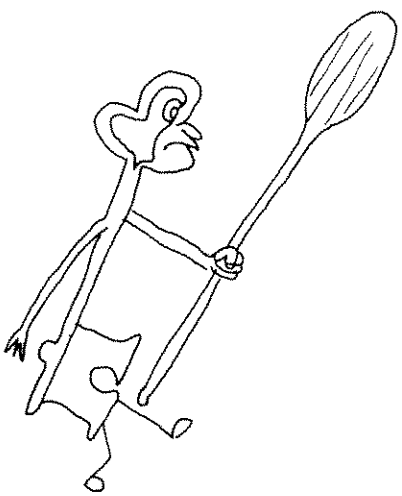
You didn't know you were already programming, did you? The keystroke sequence you developed as a solution to this equation is indeed a program.

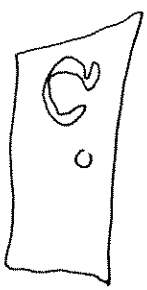
Right now, the program is recorded in your mind (and a few pages back). When you work through the sequence of keystrokes to convert a Fahrenheit temperature to Celsius, you must first call up the program stored in your mind, and with the help of your fingers, work through it, step–by–step, starting with ENTER and ending with ÷.

BUT THERE IS A BETTER WAY.

Why clutter YOUR mind with the numerous keystrokes required to solve common mathematical problems? Why not simply store those keystrokes in the continuous memory of your HP calculator?!? Afterall, that "C" in HP–11C and HP–15C stands for "Continuous Memory," so why not discover the beauty of the "C"?







Try this: Key in a program that you can use to solve the equation:

$$\frac{5 \times ((\text{INPUT}) - 32)}{9} = (\text{OUTPUT})$$

Before you can find the solution to this, you need to take a look at the program memory of your calculator. This memory is where you can store keystroke sequences (i.e. programs) for repeated use.

The program memory of the HP–11C is slightly different from that of the HP–15C, so there are two separate sections on this subject. If you have an HP–11C, you will want to read the upcoming section, but if you have an HP–15C, please turn to page 71.

PROGRAM MEMORY IN THE HP–11C

Program memory in the HP–11C is easy to understand.

First, you have 63 lines of memory to be used only for programming (a "line" is a step in a keystroke sequence, such as [ENTER], [STO] 1, [+], etc.).

Then, once you have used those 63 lines, if you need more lines, the calculator starts automatically to convert data registers to "blank" program lines––as you need them! Each data register will be converted into 7 lines of program memory.

The first register to be converted is register .9, the second is .8, etc. If you multiply the twenty available registers by seven lines and add those first "free" sixty–three lines, you will find that the longest program that will fit in your calculator is 203 lines.

It's important to remember that your calculator automatically converts storage registers to program memory, because once a register is converted, it is no longer available for storing a number.

For example, if you have a program consisting of more than 63 lines stored in your calculator, and you try to store a number in register .9, you will get an ERROR 3 message in your display. Storage register .9 no longer exists! It will come back only when you clear program memory or reduce the length of the program to 63 lines or less. OK?

Now, you may be wondering how many keystrokes can fit in one program line. Well...that depends. Generally, a single operation will use one program line. Therefore, ⊞ would be a program line, but f FIX 9 would use just one line, also.

PROGRAM MODE

Now, how do you store program steps? How do you record those keystroke sequences for repeated use?

Well, when you turn on your calculator, it is in "Run mode." You can put your machine into "Program mode" by pressing [g] [P/R] (Program/Run). Try it.

Just as the [ON] key is used both for turning the calculator on and turning it off, the [P/R] key is what you use to switch back and forth between Program mode and Run mode.

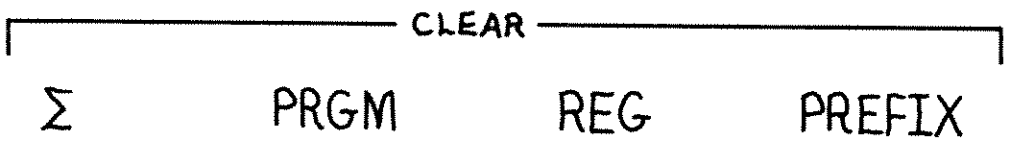
Now you should see the little word PRGM (the program annunciator) appear in the lower right-hand corner of the display. This program annunciator tells you that your calculator is in Program mode.

Look at the display while your calculator is in Program mode. You see a number at the left (this is called the line number), followed by a dash and perhaps more numbers.

The first thing you need to do is clear the program memory of your calculator. Press f CLEAR PRGM.

Did you find the CLEAR PRGM key? Don't let that bracket notation fool you.

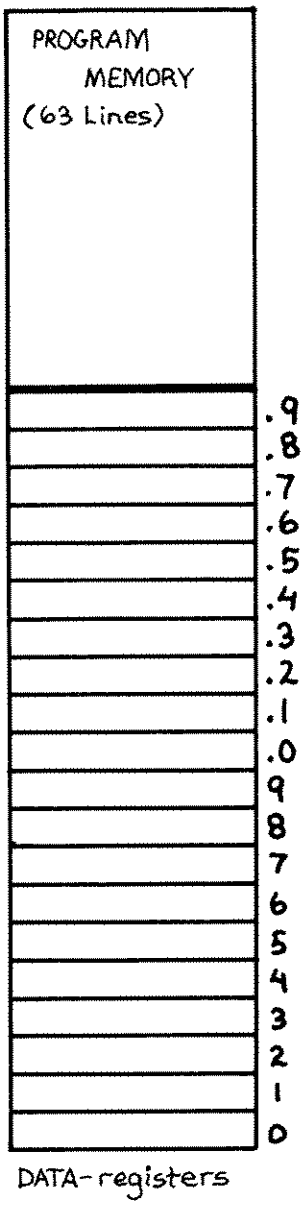
This:



means this:

When you press f CLEAR PRGM--with your calculator in program mode--this clears away any programs that you had stored there previously.

Now that you have cleared program memory, this picture describes the memory of your calculator:



Each DATA-register can be converted into 7 lines of program memory.

THE MEM FUNCTION

The HP–11C has a function called MEM (MEMory) that gives you a description of the status of its memory. Press g MEM and HOLD DOWN the MEM key. As long as you hold down this key, the calculator will display a message like "p–63 r–.9."

This means you have 63 lines available for programming (p), and the highest data storage register (r) available is register .9 (register 19). So you have 20 data registers (0 through .9) and 63 blank program lines. Your calculator's program memory is a "clean slate," which makes sense, because you just pressed a key to clear away (erase) all programs.

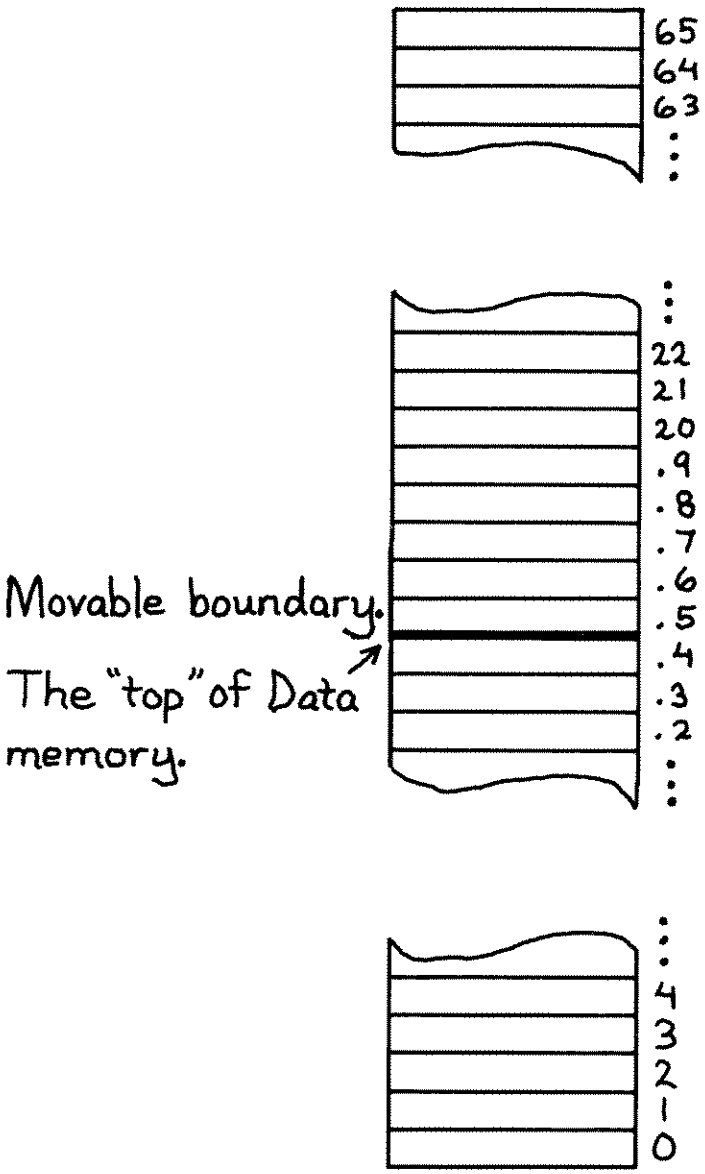
Now zip on ahead to page 82 (unless, of course, you'd like to derive additional enjoyment from learning about the program memory of the HP–15C).

PROGRAM MEMORY OF THE HP–15C

Similar to us humans, the HP–15C is endowed with a certain amount of memory. The HP–15C can store lots of numbers and programming information and if you clearly understand how it is storing this information, you can make the best use of this powerful little machine.

The human memory is very complex, and fortunately, in using it we don't have to know how it works (or doesn't work). But the HP–15C is a little pickier. If you find your calculator frequently flashing ERROR 3, ERROR 4 or ERROR 10 at you, then you need to have a clearer picture of how it uses its memory.

So, how much memory does your HP–15C have? That's an easy question to answer––the HP–15C has 66 registers of memory––but that answer doesn't tell you much, does it? This picture may help:



Back on page 5, you saw a picture of your calculator's data registers, and you learned that the number of data registers in your calculator could vary. In fact, YOU CONTROL THE NUMBER OF DATA REGISTERS IN YOUR CALCULATOR.

Basically, you set the number of data registers and then use the remainder of memory for storing things, like programs, matrices, an imaginary stack, etc. That's all there is to it. By setting the number of data registers, you are moving a boundary that separates "data register memory" from "memory used to store other things."

If you decide to use all of the calculator's memory for data registers and you move the boundary to the top of register 65, then you will have 66 data registers (0 through 65). But if you do this, you will have no memory to store programs. So, when you try to key in a program, or key in a matrix, or perform any other operation that makes use of "memory used to store other things," your calculator will scream: ERROR 10, and you will probably blush (ever so slightly).

Similarly, if you move the boundary down to the top of register 6, as shown in this picture, and then try to store a number in register 12 or 15, you will generate another error (ERROR 3), accompanied undoubtedly by more blushing.

Every register that is located above the data register boundary can hold seven lines of program memory. The lowest you can move the boundary to is the top of register 1. If you did this, the only data registers you would have available would be registers 0 and 1. But you would have 64 registers of "memory used to store other things." Now, 64 times 7 is 448. That means that if you didn't have anything else stored in memory, you could fit 448 program lines into your calculator.

(Could this be rivaling the memory we humans possess...?)

PROGRAM MODE

Now, how do you actually store program steps? How do you record those keystroke sequences for repeated use?

Well, when you turn on your calculator, it is in "Run mode." You can put your machine into "Program mode" by pressing [g] [P/R] (Program/Run). Try it.

Just as the [ON] key is used both for turning the calculator on and off, the [P/R] key is what you use to switch back and forth between Program mode and Run mode.

Now you should see the little word PRGM (the program annunciator) appear in the lower right-hand corner of the display. This program annunciator tells you that your calculator is now in Program mode.

Look at the display while your calculator is in PROGRAM mode. You see a number at the left (this is called the line number), followed by a dash and perhaps more numbers.

The first thing you need to do is clear your memory (that is, clear the program memory of the calculator). This is necessary because the discussion on the following pages assumes that your calculator's memory has been cleared.

Press f CLEAR PRGM.

Did you find the CLEAR PRGM key? Don't let that bracket notation fool you.

This:

means this:

When you press f CLEAR PRGM with your calculator in PROGRAM mode, any programs that you had previously keyed in are cleared away. Now get back into RUN mode by pressing g P/R. The little program annunciator disappears and the number in the X–register comes into the display. Finish clearing your calculator's memory by pressing
f MATRIX 0 g CF 8

(Later on, you'll learn what that does.)

MOVING THE DATA REGISTER BOUNDARY

With your memory clear, you can freely move that boundary between data registers and "memory for other things."

Try this: Set your calculator so that the highest numbered data register is register 65.

Solution: 65 f DIM (i)

(If you know all about the DIM (i) function, and the RCL DIM (i) function, then flip on over to page 82.)

The DIM (dimension) (i) function is what you use to move that boundary between "data register memory" and "memory used to store other things."

When you press f DIM (i), the calculator moves that boundary to (the top of) the register specified in the X–register. The calculator wouldn't do this, however, if in doing so it would destroy any information currently stored in the "memory used to store other things." That memory is given top priority by the machine.

But since you have just cleared the entire memory, the calculator knows you have nothing in there to lose, so by pressing 65 [f] [DIM] [(i)], you have moved that boundary to the absolute top of your calculator's memory. You have provided the maximum number of data registers. Pretty slick, eh?

To prove this, put your calculator into program mode ([g] [P/R]) and try to key in a program line (e.g., press [SIN] or [1/x]). The calculator will show ERROR 4. This means that you have no room allocated to store programs. Clear the ERROR message by pressing any key, and then get out of program mode ([g] [P/R]).

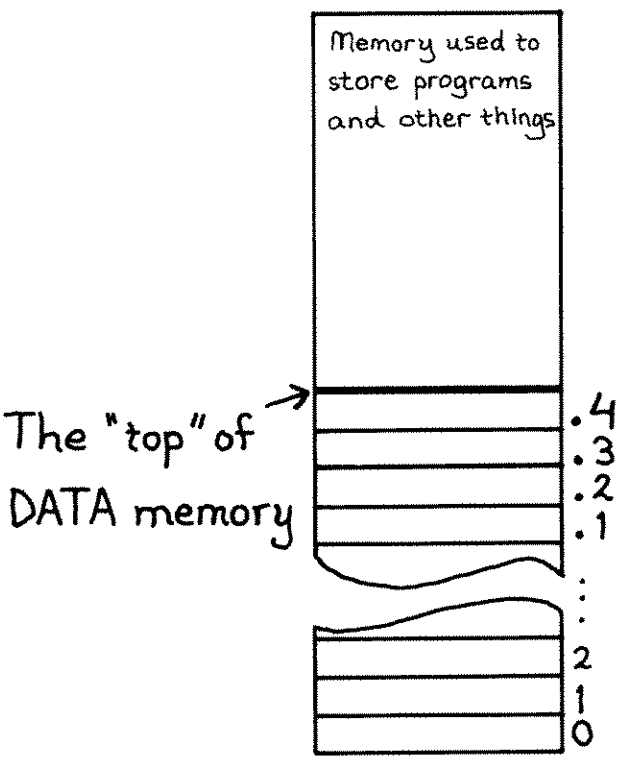
Now suppose you want to try to store a number in register 65 or register 50 just to prove to yourself that they exist (do you doubt?). But you can't just key in 3.1416 [STO] 65 (try it)! The calculator will take that as 3.1416 [STO] 6, and then the 5 will go into the X–register as the first digit of a new number. There are new and different procedures for using registers 20–65.

So for the moment, just take for granted that those registers do indeed exist. You'll learn how to use them in the section starting on page 144.

Try this: Set your calculator so that the highest numbered data register is register .4.

Solution: 14 [f] [DIM] [(i)]

Remember back on page 6, we said that register .4 was the same as register 14, that register .7 was the same as register 17, etc.? When you key in 14 [f] [DIM] [(i)], the calculator moves the boundary to the top of register 14 (which is called register .4). Your memory looks like this:



Now, try to store a number in register .9. Press 3.1416 STO .9. The calculator displays ERROR 3, and you blush--slightly. That means data register .9 (register 19) does not exist. The highest numbered data register is 14 (.4). Again, clear the ERROR message by pressing any key. Now try STO .4. It works. But try STO .5. You get ERROR 3, and more blushing. It's obvious where the boundary is, right?

Try this: Set your calculator so that the highest numbered data register is register .9.

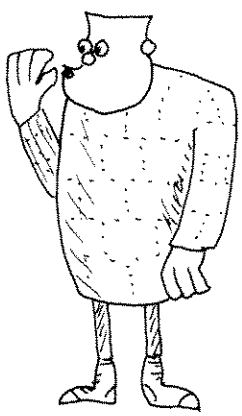
Solution: 19 f DIM (i)

THE MEM FUNCTION

The HP–15C has a handy function called MEM (MEMory) that gives you a description of the status of your memory. Press g MEM and HOLD DOWN THE MEM KEY. As long as you hold the MEM key down, the calculator will display a message such as "19 46 0–0."

The first number tells you the location of the boundary between "data register memory" and "memory used to store other things." The second number tells you how many registers of that "memory used to store other things" are empty. And the last two digits give you information about how much of that "memory for storing other things" is being used to store program lines. Isn't that thoughtful?

Another way to find out where the data register boundary is located is by pressing RCL DIM (i). If you do that now, the number 19 should come into the display. Try it!



THE PROBLEM AT HAND

Where were we?...Oh yes, temperature conversions.

The side trip you took through the details of program memory should have left you with an empty program memory and 20 data registers (0 -- .9).

Now that you know you have enough memory to do some programming, it's time to look at the solution to that temperature conversion problem.

Here are the keystrokes to create a program to solve the equation:

$$\frac{5 \times ((\text{INPUT}) - 32)}{9} = (\text{OUTPUT}).$$

Solution: [g] [P/R] 32 [−] 5 [×] 9 [÷] [g] [P/R]

(If this is too easy for all of you calculator wizards, go ahead to page 85.)

When you're going through those keystrokes, the numbers that come up in the display may not make much sense. They are called "keycodes" and are described in detail later on (page 91), so don't worry about them right now.

Here's a brief description of what's happening as you press the keystrokes of that last solution:

The keystrokes [g] [P/R] switch your calculator from RUN mode to PROGRAM mode.

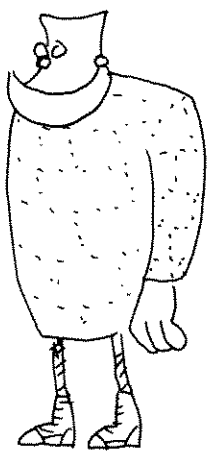
If you have been following along up to this page, your display will show "000–" with the program annunciator (PRGM) showing in the lower right hand corner of the display. "000–" means that the program "pointer" is on line 000 of program memory. This program pointer is simply a little counting device that tells the calculator what line of a program it should look at (or perform) next.

As you key in the program (32 [−] 5 [X] 9 [÷]) you will see the line number change. The program pointer is moving!

(If you should make a mistake, just press [f] CLEAR [PRGM] and start again by keying in 32.)

After you press [÷] the display will show "007– 10" indicating that there are 7 lines to this program.

Finally, get out of PROGRAM mode by pressing [g] [P/R].



RUNNING THE PROGRAM (R/S)

When you switched back to run mode, the program pointer was positioned to the last line in the program. You are going to run this program now, but first you need to move to the top of the program. Press g RTN to move the pointer up to line 000.

Now try a temperature conversion.

Try this: Convert 212 degrees Fahrenheit to degrees Celsius.

Solution: 212 R/S (Answer: 100)

Try this: Convert the following Fahrenheit temperatures to Celsius temperatures:

32, 70, 85, 100

Solution: 32 R/S
70 R/S
85 R/S
100 R/S

Now think back and compare the manual solution for these temperature conversions to the same solution recorded as a program. You will probably smile smugly, and the tiny muscles in your fingers will be eternally grateful.

On page 61, you converted 97 degrees Fahrenheit to Celsius by keying in 97, pressing ENTER, then working through the keystrokes:

32 − 5 × 9 ÷

Now you have a program to do the above keystrokes; so to convert 97 degrees Fahrenheit, you would press 97 R/S.

Question: The first two steps of the recorded program are 32; that is, the program puts the number 32 into the X–register. But if you've just keyed in 97 before starting to run the program, why doesn't that 32 write over the 97? Or why doesn't it continue to form one number: 9732? Normally, if you want to key in two numbers in a row, you have to tell your calculator to save the first one; to do this, you press [ENTER], as in the manual solution above. So why don't you have to press 97 [ENTER] [R/S] ?

Answer: Because the [R/S] function––like most functions––ENables stack lift, so that the next number keyed in will automatically bump the previous number up to the Y–register, safe and sound. Smart machine, no?

Voilá! You have a program in memory and you can use it to convert the 50 temperatures from that physics experiment. This allows you to whip through all the calculations of this experiment in a matter of minutes. And this gives you more time to indulge in other, more pleasant aspects of life (such as the consumption of cold beer on hot days, or the continued reading of this literary classic).

MOVING AROUND IN PROGRAM MEMORY

You'll probably begin to notice how often you are thinking about the program pointer: Where is it? What's it doing? (Will it be home by dark?), etc.

Well, your calculator is always "pointing" to a line in program memory. To find out which line it's pointing to, you simply have to put the calculator into program mode and look at the display. (The line number is the number on the far left of the display.)

Before you ran your temperatures program for the first time, you had to move the calculator's program pointer to the top of the program, by pressing [g] [RTN] in RUN mode. This demonstrates two things that are always true:

1. If you run a program by pressing [R/S] the calculator starts at the line it's pointing to and works downward, executing each line in the program.

2. Pressing [g] [RTN] in RUN mode will move the program pointer to line 000.

Question: Now that you've run the program, where is the program pointer?

Answer: At the top of the program (line 000).

When your calculator reaches the end of a program, it jumps to line 000 and stops. This allows you to run your program over and over, each time keying in a new input and pressing R/S.

But notice that there are several other ways to move the program pointer around:

The SST (Single STep) function moves the pointer ahead one line. The BST (Back STep) function moves the pointer backwards one line.

And if you want to move to line "nnn" of your program (where "nnn" is some 3-digit line number--002, 156, etc.), you can key in GTO CHS "nnn" (if you have an HP-15C) or GTO . "nnn" (if you have an HP-11C).

Try this: Put your calculator into program mode and move to line 005.

Solution:

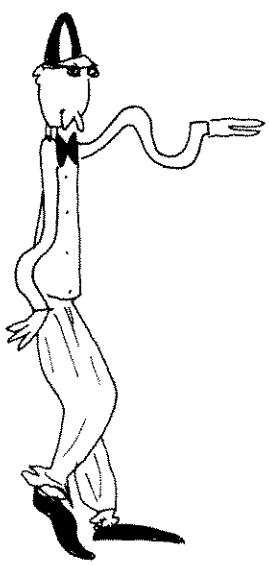
g P/R GTO CHS 005 (for HP–15C owners)

g P/R GTO . 005 (for HP–11C owners)

(In the next section, you'll learn why this function is different for each of the two calculators.)

Try this: Move backwards one line, to line 004.

Solution: g BST



KEYCODES

When you switch into program mode and look at the lines of a program, the display shows merely a series of numbers (called keycodes, as you may recall). Now, to relieve the suspense, look at what those numbers mean:

As you already know, the number at the left in the display is the program line number. That is, everything to the left of the hyphen is the line number.

Everything to the right of the hyphen is a coded description of the keys you pressed to make up that program line--plain and simple.

Most of the keys are described by two digits which indicate their row (counted from the top of the keyboard down) and column position on the keyboard (counted from left to right). For example, the keycode for the [CHS] key is 16 (row 1, column 6), the [X<>Y] key is described by 34, etc.

But the number keys ([0] through [9]) are described by only one digit--the digit that they represent. For example, the keycode for [5] is 5, [6] is 6, etc. This isn't so bad, is it?

Question: What is the keycode for the [EEX] key?

Answer: 26

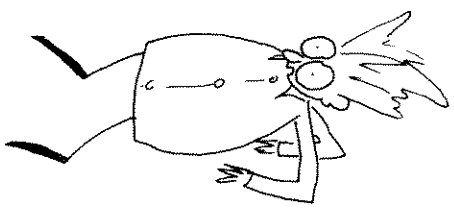
Question: What is the keycode for the [X] key?

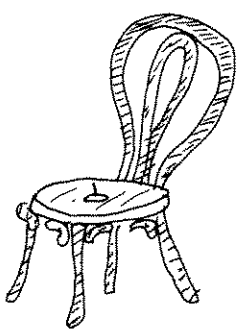
Answer: 20 (row 2, column 10--called column 0)

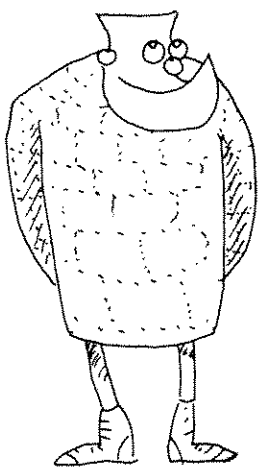
Question: How would the program line [STO] 5 be represented in the display?

Answer: nnn- 44 5 (nnn is the line number)

Do you feel somewhat comfortable with keycodes? If not, don't worry; you'll get used to them.







REVIEW

Once again, it's time to distill things down to essentials:

In this chapter you've learned a little about program memory in your calculator.

You've learned that keying in a program to solve some problem is almost identical to using manual keystrokes to solve that same problem.

You've learned the difference between RUN mode and PROGRAM mode; what the PRGM annunciator means in the display; and how to move around in a program by using the SST, BST, and GTO functions.

Try this quiz just to check yourself to see how much you've absorbed.

QUICK QUIZ

1. What's the difference between RUN mode and PROGRAM mode? Can you tell which mode your calculator is switched to by looking at the display?

2. What keys do you press to clear all the programs from program memory?

3. What do the numbers mean that come into the display when you press [g] [MEM]?

4. If the program pointer is pointing to line 005 in a program, and you press [R/S] (in Run mode), the program will begin to run at line_____.

   If you press [g] [RTN] [R/S], the program will begin to run at line_____.

QUICK ANSWERS

1. In RUN mode, functions will be executed immediately when you press the keys. In PROGRAM mode, functions are stored as lines of a program (to be executed when the program is run). If the calculator is in PROGRAM mode, the little PRGM annunciator will appear in the display.

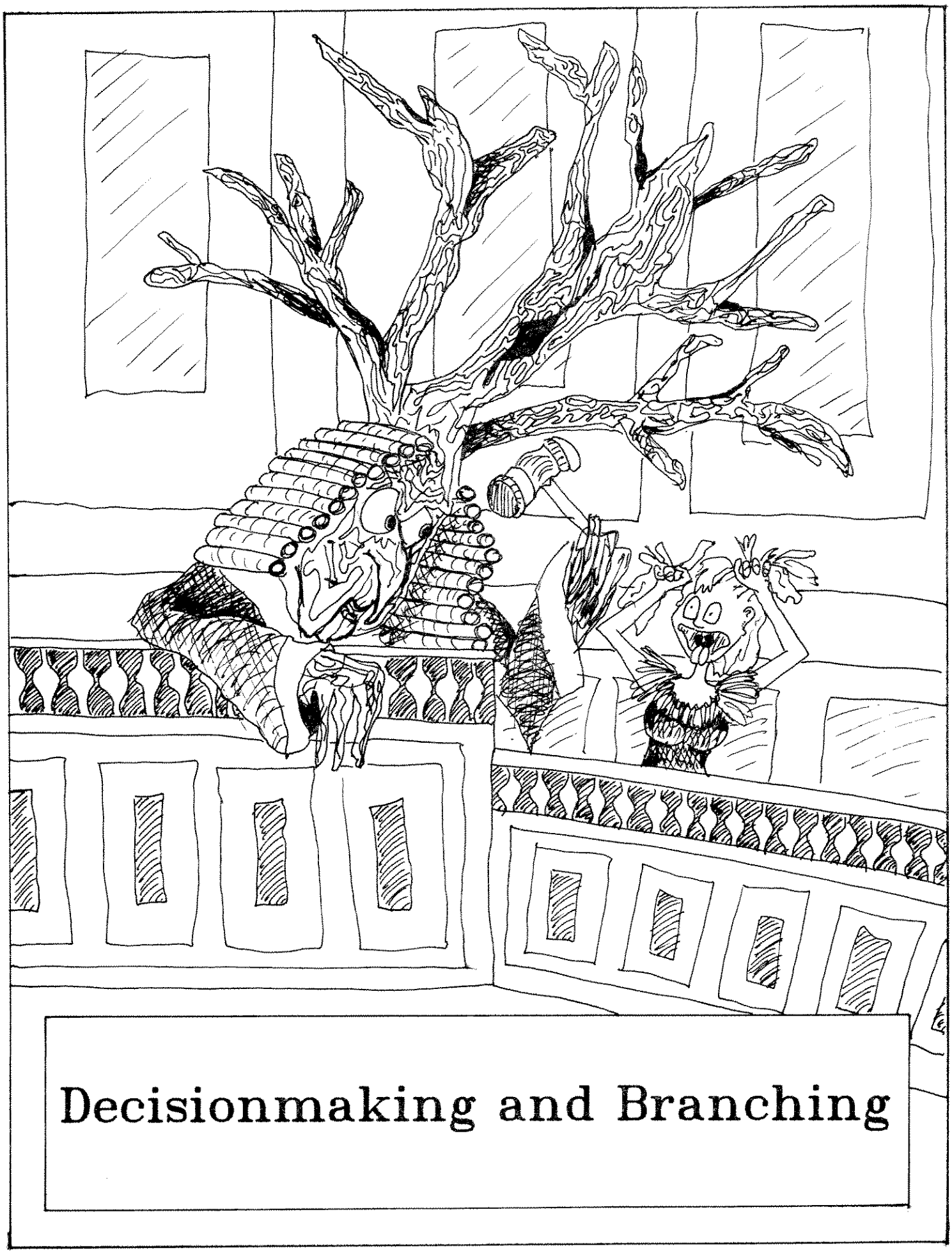
2. g P/R (to switch to Program mode)
   f CLEAR PRGM (to clear all programs)

3. On the HP–11C: The number following "p" is the amount of empty program lines in the first 63 lines; the number following "r" is the highest numbered data register available for storage.

   On the HP–15C: The leftmost number is the highest numbered data register available for data storage; the center number is the number of empty registers in the "memory for other things;" and the rightmost (hyphenated) digits tell you how much of the "memory for other things" currently contains program lines.

4. Line 005; line 000.

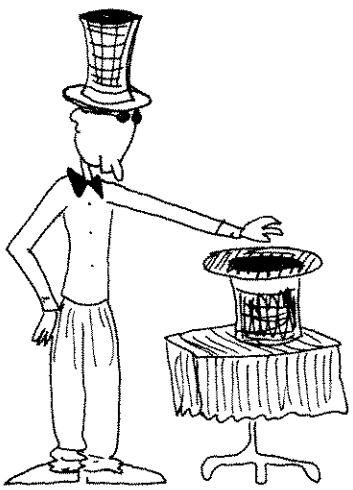
Notes



Decisionmaking and Branching

DECISIONMAKING AND BRANCHING

Up to this point, you've seen only what we call "a naked program" (a program that is always run from top to bottom, with no frills, made up mainly of arithmetic functions). Now it's time to look at some of the niceties that your calculator is equipped with--some functions that you can use to create fairly elaborate programs.



Try this: Create and run a program that uses the display to count by ones, starting at 1. It should first display 1 for a moment, then 2, 3, 4, etc., up to 25. Then, if flag 1 is set and the value in register 0 is greater than the value in register 3, the program should finish with PI in the X–register (rounded to 3 decimal places); otherwise, it should finish with 0 in the X–register.

Solution: The solution to this appears on page 140. If, without looking at the solution, you can write a program that does all this, then GTO page 138 and continue on from there. If not, then you should probably continue here and read about labels.

LABELS LBL

Labels are used in programs as markers. They can mark the top of a program, or they can mark important "landing points" for jumping within a program. You can tell the calculator to move its program pointer to a label either with direct (i.e. manual) keystrokes or with "jumping" instructions that you have recorded as program lines.

The HP–11C is equipped with the labels 0 through 9 and labels A, B, C, D, and E. The HP–15C has all the labels that the HP–11C has, plus the labels .0 through .9. (This is why you must use GTO CHS "nnn" if you want to move the pointer of your HP–15C to line "nnn." As you can see now, if you pressed GTO . "n," your HP–15C would understand the instruction to be a program line which would say, in effect, "go to LBL 'n'.")

On either machine, labels A, B, C, D, and E are named differently because they can be "called" (that is, you can run a program with them) conveniently from the keys on the keyboard.

As an example of using a label, put your calculator into program mode, and with line 000 showing in the display, press f LBL A. Assuming that the temperature conversion program from the last chapter is still intact, this puts label A at the top of that program. (If you've changed the program, you can key it in again after first clearing your program memory. See page 82.)

The program now looks like this:

(In this book, we list programs in words and symbols--rather than keycodes--so it's easier to see what each line means. Of course, your calculator will show you only keycodes.)

```
001   LBL A
002   3
003   2
004   −
005   5
006   X
007   9
008   ÷
```

The program line "LBL A" was inserted at the top of the program. The rest of the program was pushed down one line and renumbered. Your temperature conversions program is now marked by LBL A at the top.

Now take the calculator out of program mode:
g P/R.

Try this: Convert 150 degrees Fahrenheit to degrees Celsius.

Solution: 150 f A

Because you put LBL A at the top of the temperature conversion program, pressing the A key now tells the calculator to run that program, beginning at that label.

GTO, GSB, and RTN

Well, now you know how to "call" a letter-label from the keyboard. But what about the other labels? How else can you tell your calculator to find a certain label and to start running a program at that point?

The GTO ("Go TO") and GSB ("Go SuB") functions are both used for branching to a LBL (Label) in a program. When the machine encounters either of these functions in a program, it is being told something very specific.

GTO is used for branching to a LBL when you have no intention of coming back. GTO is a command that the calculator understands as "go to another part of this program and forget where you came from--just continue from there."

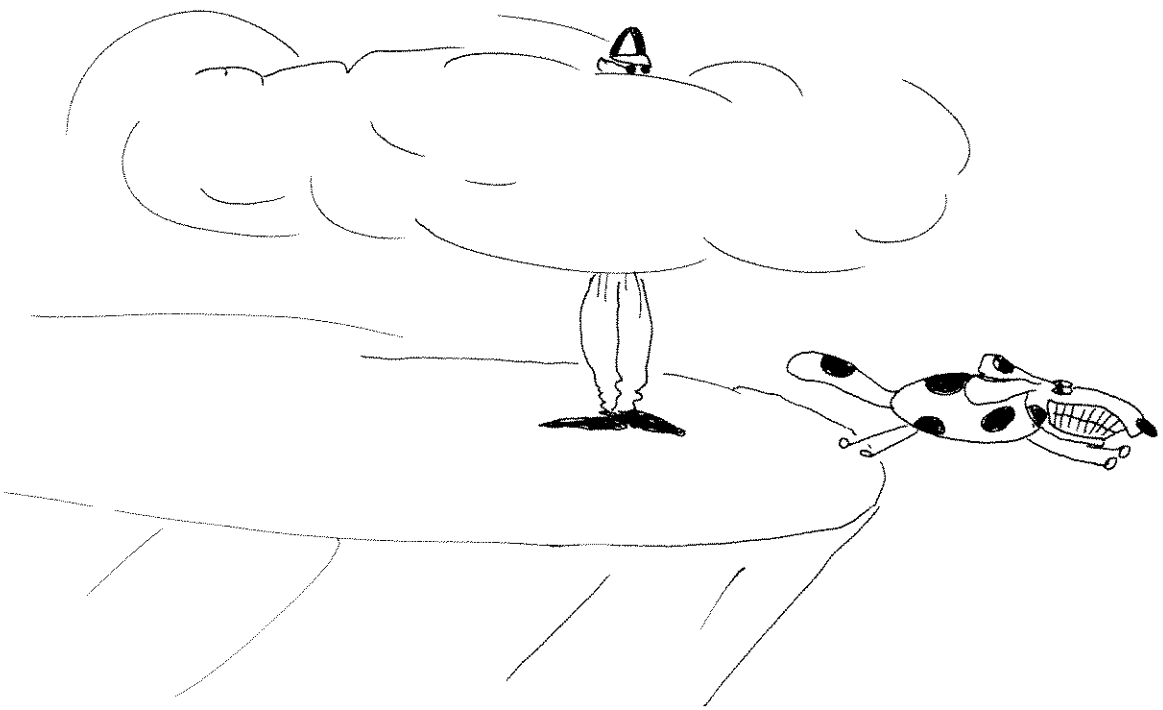
GSB, on the other hand, is used for branching to subroutines. In other words, it is used for "taking a side trip" to a LBL when you have every intention of coming back. GSB is a command that the calculator understands as: "Remember this spot and go to another part of this program; continue from there until you come to a RTN (ReTurN) statement; then come right back and continue from here."

That RTN statement is very important! The calculator plays by certain rules when it encounters a RTN in a program, and it does one of two things:

1. If the program pointer has branched off onto a "side trip"--because it encountered a GSB statement--then you might say that the GSB is "waiting" for the pointer to return from its "side trip." If this is the case--if a GSB statement is indeed "waiting"--then the RTN means: "Return to the statement following the waiting GSB."

2. If there's no waiting GSB, then the RTN statement means "STOP!"

OK? Try a few problems to clear up the fog ---------->

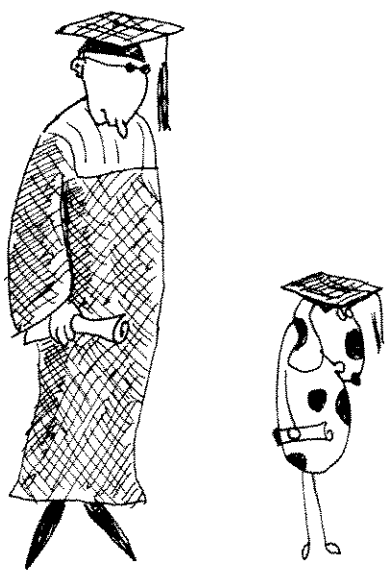


Try this: Key in a program that makes your calculator count in the display. It should start by displaying 1 for a moment, then 2, 3, 4, etc. (a foolish application, to be sure, but it demonstrates some finer points).

Solution:

```
001   LBL B
002   0
003   LBL 4
004   1
005   +
006   PSE
007   GTO 4
```

Did you key in this program and run it without looking at the solution? If so, you are now eligible to graduate to page 108. Otherwise...——————>



PROGRAM LOOPS

Look at this last solution.

Lines 003 through 007 make up a "program loop." The LBL 4 statement is the top of the loop, and GTO 4 is the bottom.

The body of the loop (lines 004, 005, and 006) is a procedure that merely adds 1 to the contents of the X–register and momentarily displays it (line 006 is the PauSE function).

Notice how the program puts a 0 into the X–register at step 002––before it begins to go around the loop.

Finally, when the program pointer gets to the line GTO 4, the calculator looks for LBL 4 and then "jumps" to it. Thus the program will continuously go around the loop (adding 1 to the value in the X–register and displaying the result) until it is stopped––when you press R/S. Make sense?

The keystrokes for this program are:

KEYSTROKES	DISPLAY	
g P/R f PRGM	000–	
f LBL B	001–	42, 21, 12
0	002–	0
f LBL 4	003–	42, 21, 4
1	004–	1
+	005–	40
f PSE	006–	42 31
GTO 4	007–	22 4
g P/R	(Normal numerical display)	

Now, run this program by pressing f B (why do you press these keys? See page 102 for a reminder). The program works! (If your calculator isn't counting, then repeat the above keystrokes.)

To stop the program, just press R/S.

Try this: Using three GSB statements, write a program that counts--in the display--from one to three.

Solution:

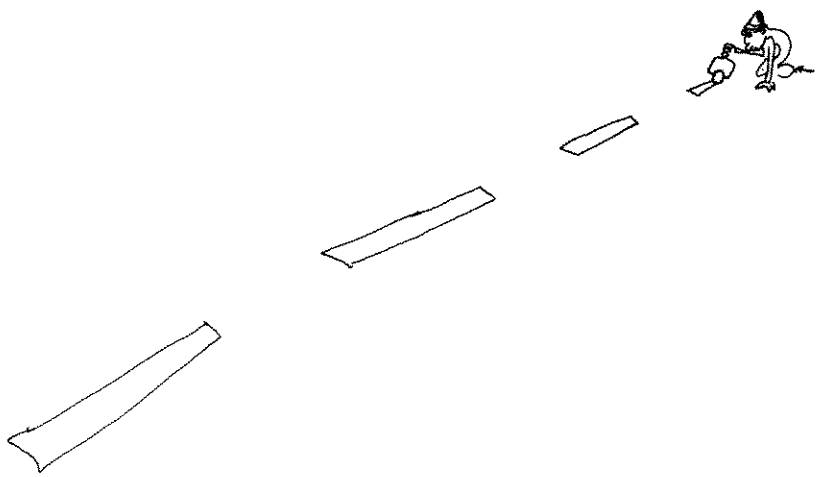
```
001   LBL B
002   0
003   GSB 4
004   GSB 4
005   GSB 4
006   R/S
007   LBL 4
008   1
009   +
010   PSE
011   RTN
```

(If you knew the easy way to do this--by editing the existing program--then try page 112.)

EDITING A PROGRAM

As you may have noticed, this program differs only slightly from the one you keyed in on the previous page. To change the previous program to this one, all you need to do is insert 4 lines (three GSB 4's and a R/S) after line 002, and then change the last line in the program from GTO 4 to RTN. Here's how you would do this:

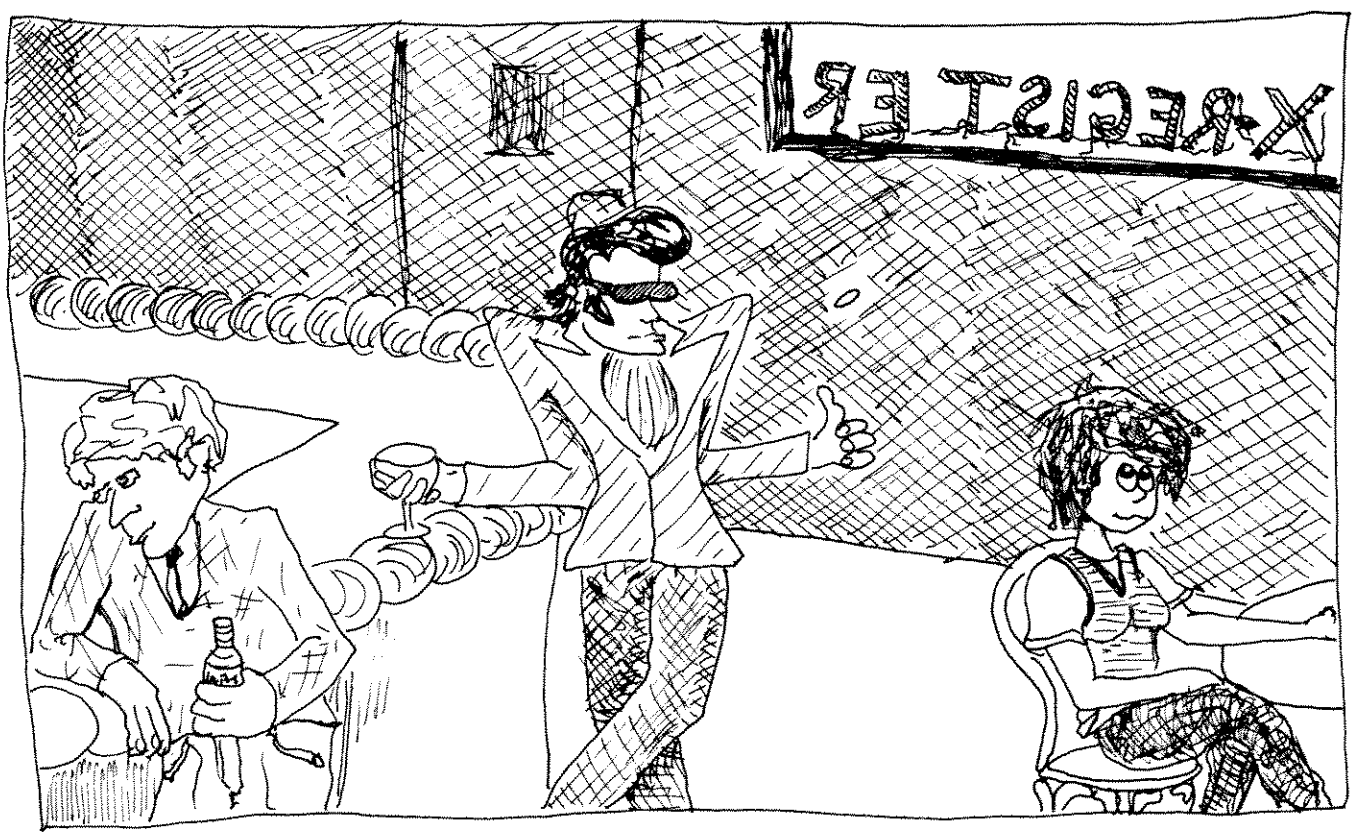
Switch into PROGRAM mode (g P/R). Now, if you have an HP–11C, press GTO . 002 (but if you have an HP–15C, press GTO CHS 002). This moves the program pointer to line 002 of the program. Press GSB 4 three times; then press R/S (you'll end up at line 006). This inserts the necessary lines after line 002.



Now you have to change the last line in the program from GTO 4 to RTN. First, move to the top of the program (if you have an HP–11C, press GTO . 000; if you have an HP–15C, press GTO CHS 000).

From the top of the program you can move to the last line in the program by pressing g BST. In other words, if you're at the top of the program, you can "wrap around" to the bottom (the end) by Back–STepping.

Once you're at the last line of the program (line 11), you can delete the GTO 4 statement by pressing the backarrow ← key. Finally, press g RTN.
Now you have the new program! Press SST to move up to line 000 of this program.



If you want to, you can [SST] (single-step) through your entire program to verify that you have it correct. This is what you should see:

KEYSTROKES	DISPLAY		EXPLANATION	
[SST]	001-	42, 21, 12	001	LBL B
[SST]	002-	0	002	0
[SST]	003-	32 4	003	GSB 4
[SST]	004-	32 4	004	GSB 4
[SST]	005-	32 4	005	GSB 4
[SST]	006-	31	006	R/S
[SST]	007-	42, 21, 4	007	LBL 4
[SST]	008-	1	008	1
[SST]	009-	40	009	+
[SST]	010-	42 31	010	PSE
[SST]	011-	43 32	011	RTN

Now get out of program mode ([g] [P/R]) and press [f] [B]. Again, it works! You've taught your calculator to count to three (Whoopee)!

Can you see how GSB is working? The GSB 4 at line 003 sends the program pointer to LBL 4, where 1 is added to the X-register, and the result is displayed. The RTN at line 011 sends the program pointer back to the line immediately following that GSB 4 (i.e. back to line 004).

This line (line 004) is another GSB 4 statement, which sends the program pointer back to LBL 4. The value in the X–register is incremented, displayed, and the RTN statement sends the program pointer back to the line immediately following the "waiting" GSB 4 (i.e. back to line 005).

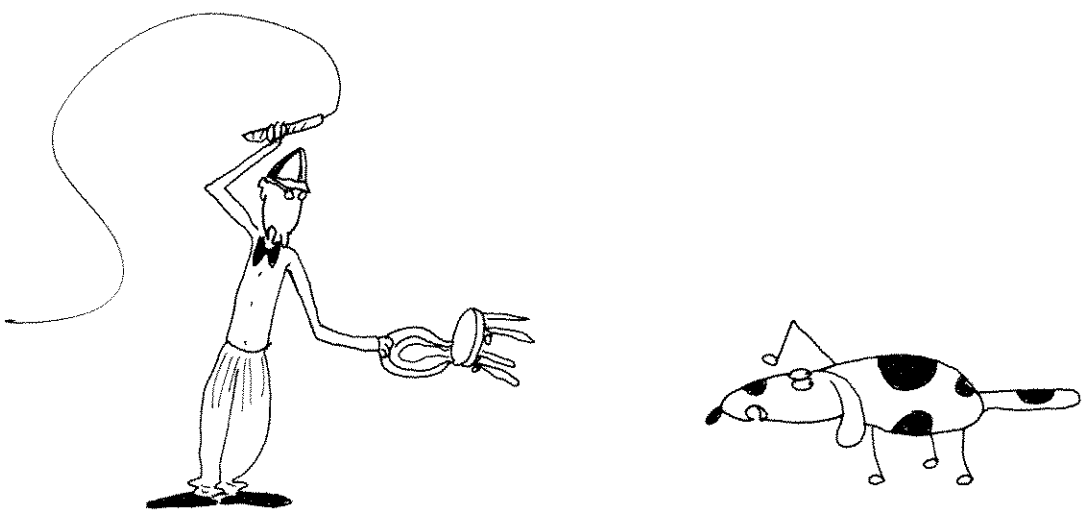
Again, (line 005) is another GSB 4 statement, and this time, when the program pointer RTNs to the statement immediately following line 005, it finds a R/S statement, which causes the program to STOP.

Question: What would happen if you used a RTN at line 006 instead of a R/S?

Answer: The program would run exactly the same. Unless there is a "waiting" GSB for the pointer to return to (and in this case there wouldn't be), the RTN statement means STOP.

After you run the program, if you put your calculator into PROGRAM mode by pressing g P/R, you would see that the program pointer is waiting at line 007 of the program. The program was stopped by the R/S at line 006.

As you may have guessed, the program lines between LBL 4 and RTN form a subroutine. The program is counting to three by simply "taking a side trip"—three times—to a subroutine that causes the calculator to add 1 to the X–register and then display the result. This gives the effect of counting to three.



CONDITIONAL TESTING

All right, now you know how to make your calculator count to three. But how could you make that program decide to stop at ANY number you choose? Surely you don't want to use 10 or 20 or 100 consecutive GSB statements, do you? No, not really (for one thing, the sheer monotony of keying in all the GSB's would put you to sleep).

Look back for a minute at that first counting program--the one that simply keeps counting until you press R/S or turn the machine off (or the battery runs low):

```
001   LBL B
002   0
003   LBL 4
004   1
005   +
006   PSE
007   GTO 4
```

Try this: Modify the counting program so that it counts from 1 to 10 and stops.

Solution:

```
001   LBL B
002   9
003   ENTER
004   0
005   LBL 4
006   1
007   +
008   PSE
009   X≤ Y
010   GTO 04
```

(If you see how this program works, go to page 123.)

To breeze through the above problem, you have to be comfortable with:

A. Moving around in program memory and editing a program.

B. Conditional statements.

First, we'll look at the conditional line in this program (line 009); then we'll look at the specific key-strokes you can use to change a program that counts "forever" into a program that counts from 1 to 10.

CONDITIONAL TESTS

The HP–11C has 8 different statements that are tests (plus one flag test that we will discuss later). These 8 tests appear as gold and blue functions on the four keys in the right–hand column of keys.

The HP–15C has 12 such tests available (plus one flag test that we will discuss later). There are only two conditional tests on the keyboard: $X \leqslant Y$ and $X=0$.

But there is a function called TEST, which allows you to choose any of the other 10 tests by number (TEST 0 through TEST 9)––similar to the way you choose registers for STO and RCL.

On the back of the HP–15C, there is a table that shows TEST 0 through TEST 9. TEST 1, for example, is $X>0$, and TEST 8 is $X<Y$.

But just what IS a conditional test, anyway? ------>

THE "DO IF TRUE" RULE

By using a conditional test in a program, you are posing a TRUE or FALSE question to your calculator.

If the answer to that question is FALSE, the calculator will skip the program line immediately following that test. If the answer to that conditional test is TRUE, the calculator WILL perform the statement immediately following that test. This is known as the "DO IF TRUE" rule.

The "DO IF TRUE" rule is about all you need to know to use conditional statements in a program.

In the program that counts to 10, line 009 is 009 $X \leqslant Y$. This statement is understood to be a true or false question: "True or False? The value in the X–register is less than or equal to the value in the Y–register."

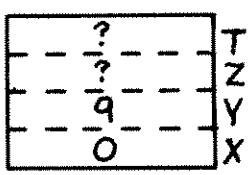
If the answer is true, that is, if the number in X is less than or equal to the number in Y, then the calculator will execute the GTO 04 statement at line 010, which sends it to LBL 04 to continue looping. But if the answer is false (if the number in X is greater than the number in Y), then the calculator skips line 010 and goes right to line 011, which ends the program.

(If you understand this program now, go to page 123.)

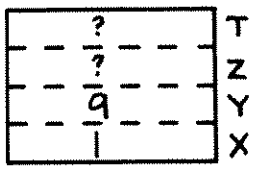
```
001   LBL B
002   9
003   ENTER
004   0
005   LBL 4
006   1
007   +
008   PSE
009   X≤Y
010   GTO 04
```

Once this program is in your calculator, you can execute the program by pressing f B (when the calculator is in RUN mode).

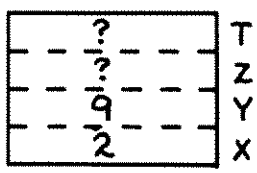
The calculator will begin to run the program at LBL B. Then Lines 002 through 004 set up the stack like this:



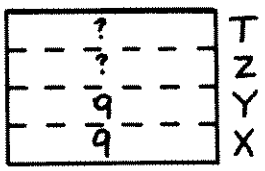
Line 005 begins the counting loop (LBL 4). On the first time through this loop, lines 006 and 007 add 1 to the X–register, and line 008 momentarily displays the X–register. Then the stack looks like this:



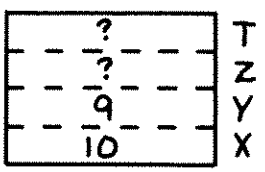
Line 009 asks the True–or False question: "Is the number in the X–register less than or equal to the number in the Y–register?" Since it is, the program continues with GTO 04. On the second time through the loop, 1 will be added to the X–register, and at line 008 (PSE) the stack will look like this:



The program will continue to add one to the X–register, displaying the result, and testing the X–value against the Y–value until it reaches the tenth time through the loop. When it passes LBL 4 for the tenth time, the stack looks like this:



The X–register is incremented (1 is added), so the stack looks like this:



Now, in line 009, when the calculator is asked the question "TRUE or FALSE? X is less than or equal to Y," the answer is FALSE. Thus, line 010 is skipped and the program ends.

MORE EDITING

Let's look at how you would change this program,

```
001   LBL B
002   0
003   LBL 4
004   1
005   +
006   PSE
007   GTO 4
```

into this program:

```
001   LBL B
002   9
003   ENTER
004   0
005   LBL 4
006   1
007   +
008   PSE
009   X≤Y
010   GTO 04
```

(If you can already handle this, go to page 123.)

To change the first program into the second program, you must make the following two changes:

1. You have to insert the conditional statement "X≤Y" after line 006 in the program.

2. You have to insert the two lines "9" and "ENTER" after the first line of the program.

Assuming you haven't yet made any changes to the original counting program, here are the steps you will use to modify this program. Switch into program mode (g P/R) and move to line 006 of the program (On the HP–11C use GTO . 006 and on the HP–15C use GTO CHS 006.). Insert the conditional test "X≤ Y" by pressing the appropriate prefix key (f on the HP–11C, g on the HP–15C) and then X≤Y.

Now, move to line 001 of the program by pressing SST twice, and then press the keys 9 ENTER.

Are you getting the hang of program editing?

Switch out of program mode (g P/R) and press f B. The calculator should count to 10 and stop. Whoopee!

FLAGS

Flags are handy tools to use in programming. A flag is a kind of indicator that has only two possible values: Set or Clear (up or down, yes or no, etc.). So a flag is a good way for the calculator to make (and remember) yes–or–no decisions within programs.

There are three functions that deal with flags on your calculator: SF (Set Flag), CF (Clear Flag), and F?. If you want to set flag 0 on your calculator, you press SF 0. If you want to clear flag 0, press CF 0.

The F? function asks the calculator a true or false question similar to a conditional test: "TRUE or FALSE? Flag so–and–so is set." You would specify the flag by number: F? 0, F? 1, etc.

The HP–11C has two flags: 0 and 1.

The HP–15C has 10 flags (flags 0 through 9), but flags 8 and 9 have a special meaning to the calculator. If you set flag 9 (SF 9), the display will blink; and if you set flag 8 you put your calculator into "Complex mode" indicated by the little 'C' in the display (the Complex mode is discussed in Appendix 3). To stop the blinking display or to take your calculator out of Complex mode, press CF 9 or CF 8, respectively.

To keep things brief and clear (that is, short and simple), we will use only flag 0 in this section. Remember: to set flag 0, press SF 0. To clear flag 0, press CF 0.

Question: What does the following modification do to your "count to ten" program? What does the program do if you set flag 0 before you run it?

```
001   LBL B
002   9
003   ENTER
004   0
005   LBL 4
006   1
007   +
008   F? 0
009   1
010   F? 0
011   +
012   PSE
013   X≤Y
014   GTO 04
```

Answer: If you set flag 0 before you run the program, it will count to 10 by twos. Otherwise, there will be no change.

Did you answer the question? Can you make the modifications to the program and run it first with flag 0 clear, then with flag 0 set? If you can, go to page 128.

To make the modifications, you need to add four lines (F? 0; 1; F? 0; +) after line 7. ONLY IF FLAG 0 IS SET will these 4 lines add an additional 1 to the value in the X–register each time through the loop.

To add those lines, switch into program mode (g P/R), and move the program pointer to line 007 (on the HP–11C, press GTO . 007; on the HP–15C, press GTO CHS 007). Then press:

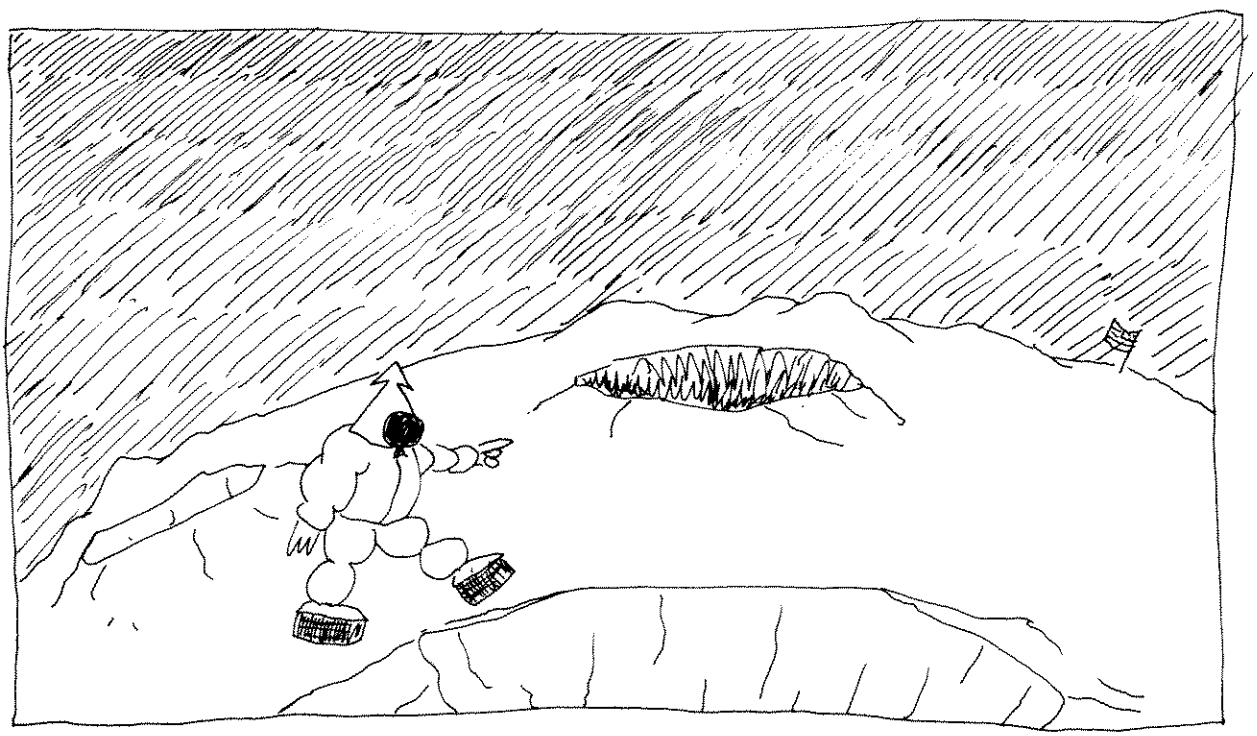
g F? 0
1
g F? 0
+
g P/R (to switch out of program mode)

Now clear flag 0 (CF 0), and run the program (f B). The calculator will simply count to ten by ones, as usual.

But try setting flag 0 before you run the program:

g SF 0
f B

The calculator counts to ten by twos (it's getting smarter)! Do you see why it's doing this? Pretend you are the program pointer, and see how those flags control whether you perform (new) steps 009 and 011.



ISG AND DSE

By now, you know all about labels and how to "call" them; you know what subroutines are; and you know how to use conditional tests. All of these different topics are somehow involved in making the program pointer jump around or skip to other lines. Now here are two other functions that also belong in that category.

The two functions are ISG ("Increment and Skip if Greater than") and DSE ("Decrement and Skip if Equal to or less than").

You would use these two functions to control program loops. They act like conditional statements, because under some conditions in a program, the calculator will skip the line immediately following either of these functions. It is this characteristic that allows you to limit the number of times a program loop is executed (as you did in the program that counts to 10).

THE CONTROL NUMBER

To help count these program loops, the ISG and DSE functions both use a number that you have stored in a register. This number is called the control number. On the HP–11C, you will always store this control number in the I–register. On the HP–15C, you may store this control number in any register.

So what does a control number look like? What is the calculator expecting?

Well, this is a control number:

1.01001

Try this: Assuming this control number is stored in the I–register, write a program that uses ISG (or ISG I on the HP–15C) to count in the display from 1 to 10 (by ones).

Solution:

```
001   LBL C
002   RCL I
003   INT
004   PSE
005   ISG (On the HP–15C use ISG I)
006   GTO C
```

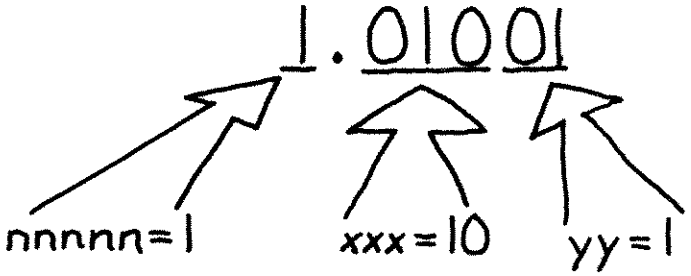
(If you understand control numbers and the above program, go to page 138.)

A control number actually represents three numbers to the calculator. If we call these three numbers nnnnn, xxx, and yy, then the calculator is looking at the control number like this:

nnnnn.xxxyy

Your calculator is concerned only with five digits on either side of the decimal place.

The number nnnnn is the current counter value, xxx is the counter test value, and yy is the increment or decrement value. For example, in the control number 1.01001, nnnnn is 1 (00001 = 1), xxx is 10 (010 = 10), and yy is 1. Look at this picture:



But what does this mean? What happens when the calculator executes the function ISG?

This is what happens:

1. The calculator adds the number "yy" to the number "nnnnn" and the result becomes the new "nnnnn." This is called "incrementing."

2. The calculator tests to see if "nnnnn" is GREATER than "xxx," and if it IS greater, the calculator skips the line immediately following the ISG statement.

Do you see where the name "Increment and Skip if Greater than" comes from? If not, reread the above two steps.

Assuming that this control number, 1.01001, is stored in the I–register, it's easy to write a program that counts (in the display) from 1 to 10. All you need is a loop that recalls the number in the I–register, then pauses to display the integer portion of that number (INT), and performs an ISG on the control number in the I–register––to test whether the loop should be repeated or exited.

Got all that? Not quite? Well, walk through it slowly and see how it works.

First, here are the keystrokes to key in the solution program––once you have switched to PRGM mode (g P/R):

KEYSTROKES

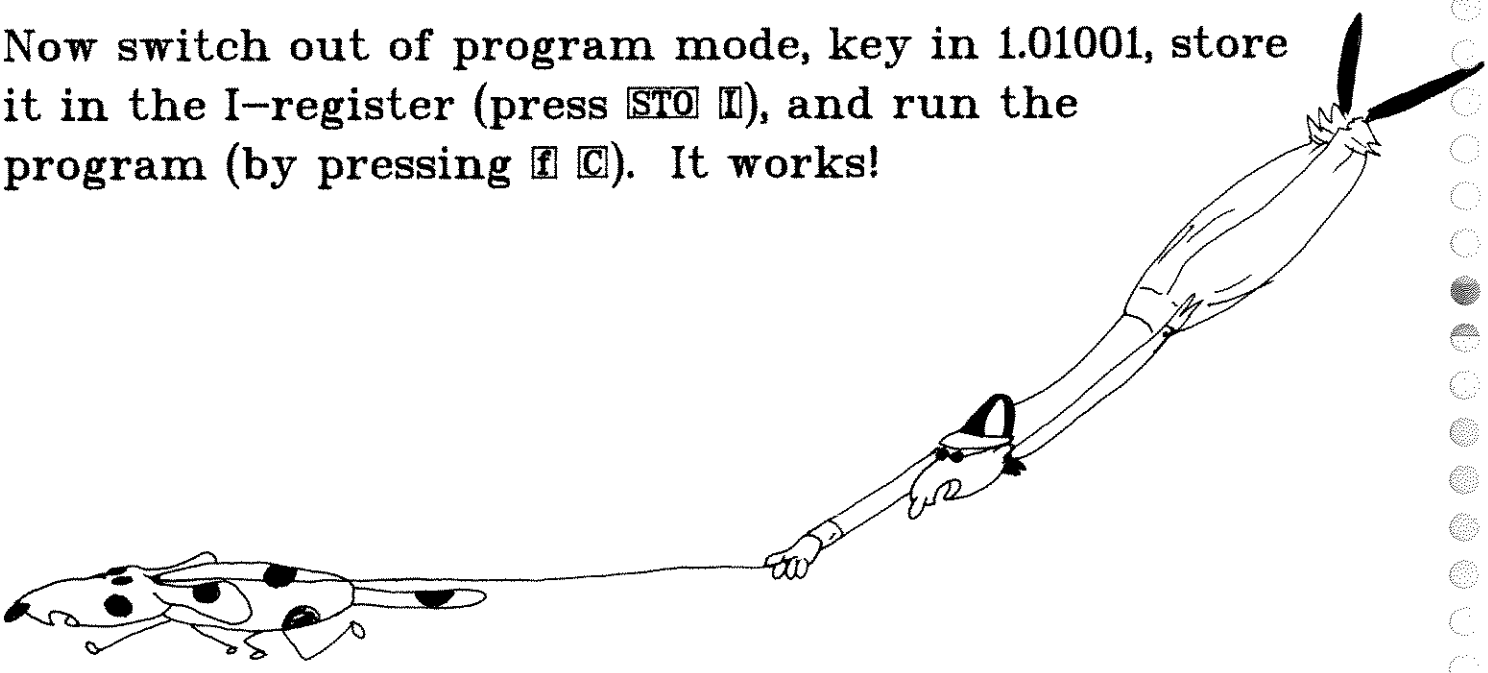
f PRGM	(to clear program memory)
f LBL C	
RCL I	(recalls the number in the I–register)
g INT	(takes the integer portion)
f PSE	(pauses a moment to display the count)
f ISG	(f ISG I if you have an HP–15C)
GTO C	

And the display shows the following, if you single step through the program using SST.

DISPLAY

```
001-   42, 21, 13
002-        45 25
003-        43 44
004-        42 31
005-        42  6  (or "005- 42, 6, 25" for the HP-15C)
006-        22 13
```

Now switch out of program mode, key in 1.01001, store it in the I-register (press STO I), and run the program (by pressing f C). It works!



```
001   LBL C
002   RCL I
003   INT
004   PSE
005   ISG (I)
006   GTO C
```

This whole program is one loop. The first time through the loop, line 002 digs 1.01001 out of the I–register, line 003 takes the integer portion of that number, which is 1, and line 004 pauses to display the 1 in the X–register. At line 005 (ISG on the HP–11C or ISG I on the HP–15C), the calculator increments the nnnnn portion of the control number by the yy portion, making nnnnn equal to 2. It then tests to see if nnnnn (2) is greater than xxx (10), and (since 2 isn't greater than 10) it does NOT skip line 006. Remember, the name is "Increment and SKIP if GREATER than," so this time it doesn't skip line 006 (GTO C); thus, the loop starts over again from LBL C.

The second time through the loop, the control number in the I–register starts as 2.01001. The third time through the loop, this number starts as 3.01001, etc.

On the tenth time through the loop, the control number starts as 10.01001. This number is recalled to the X–register and the integer portion is momentarily displayed. Finally, at line 005 the calculator increments the control number, making it 11.01001, and tests to see if nnnnn (11) is greater than xxx (10). Since 11 is usually greater than 10, the calculator skips line 006, which stops the program.

So, if you press RCL I after running the program, you'll see that the final number in there is 11.01001. (Be sure to store 1.01001 in the I–register before you run the program.)

Question: What happens if you store the number 2.01002 in the I–register before you run this program?

Answer: The program will count from 2 to 10, by twos.

The yy portion of this number is 02 (this is the increment value); the upper limit (xxx) is 010; and the beginning number (nnnnn) is 2.

Question: What number would you store in the I–register to make this program count from 0 to 500 by 50's?

Answer: .50050 (Try it.)

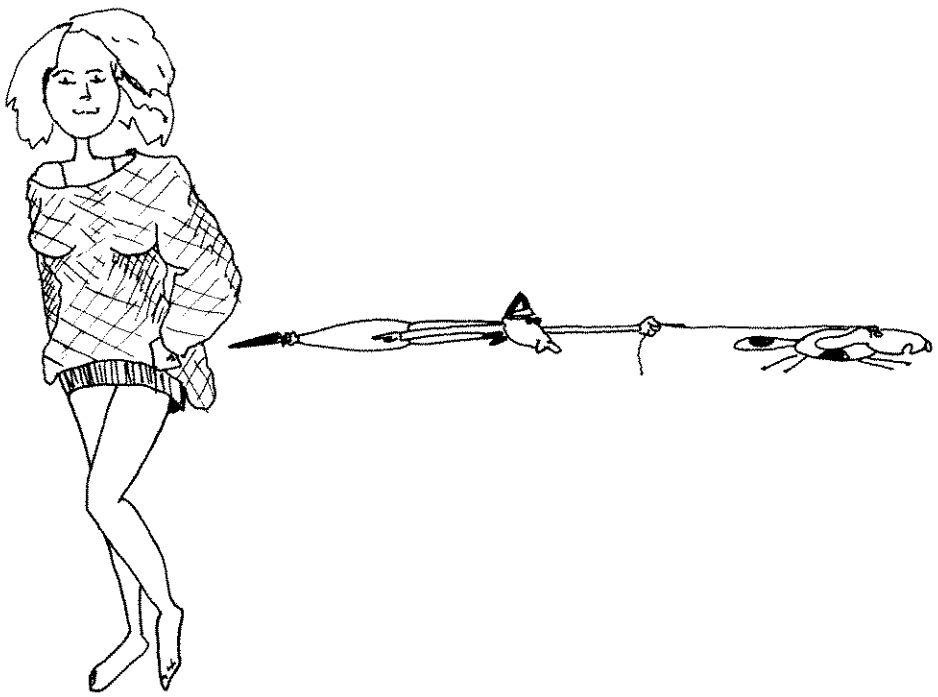
Do you see what's happening?

Question: How will this program count if you store the number 5.012 in the I–register before you run it?

Answer: It will count from 5 to 12 by ONES.

If the yy portion of the control number is 00, the calculator assumes you meant 01. The most common increment value is 1, so if there's nothing as the last two digits of the control number, the calculator increments by 1 (you can't increment by zero, right?).

Those are the basics of ISG. The other function, DSE (Decrement and Skip if Equal to or less than), works in a similar fashion--doing what its name implies.



You've been exposed to a lot of programming features in this chapter. You may not feel absolutely comfortable with the material up to now, but you'll get some practice in the upcoming sections. Now try this quiz for practice, review, and peace of mind. ----->

QUIZ

1. What are three things you can do with the ⌫ (back-arrow) key?

2. How would the program on page 108 work differently if you deleted lines 005 and 006?

3. You have the measurements for the radii of 70 different circles, and you need to find the areas of these circles. Write a "naked program" that will help you do this. Use the formula: AREA = PI × R^2.

4. Create and run a program that uses the display to count by ones, starting at 1. It should first display 1 for a moment, then 2, 3, 4, etc. up to 25, and then, if flag 1 is set and the value in register 0 is greater than the value in register 3, it will finish with PI in the X-register (rounded to 3 decimal places); otherwise, it will finish with 0 in the X-register.

ANSWERS

1. (a) Clear one digit at a time from the display (when you are in the middle of keying in a number).
   (b) Clear the X–register––just as with [g] [CLX] (i.e. when you aren't currently keying in a number).
   (c) Delete a program line (when in program mode).

2. There would be no difference.

3. 001 x^2
   002 π
   003 x

To use this program, you would simply key in a radius, press [R/S], and the result will be the area of that circle. Writing little programs like this can save you plenty of time on repetitive tasks.

4. This is also the solution to the "Try this" on page 99. This program tests your understanding of the different features that you studied in this chapter.

```
001   LBL D
002   2
003   4
004   ENTER
005   0
006   LBL 1
007   1
008   +
009   PSE
010   X≤ Y
011   GTO 1
012   RCL 3
013   RCL 0
014   CF 0
015   X>Y       (on the HP-15C press [g] [TEST] 7)
016   SF 0
017   0
018   F? 0
019   GTO 2
020   GTO 3
021   LBL 2
022   F? 1
023   π
024   LBL 3
```

(Remember, there are several different ways to write any program. The method here isn't as important as the results.)

Lines 001 through 011 of this program simply count from 1 to 25, the same as the counting programs that you saw in this chapter.

Lines 012 through 024 test to see if the following two conditions are true:

1. The value in register 0 is greater than the value in register 3.
2. Flag 1 is set.

If both of these conditions are true, then the calculator puts the PI into the X–register. But if either one of these conditions is not true, the calculator leaves a zero in the X–register.

Go through the last 12 lines in the program a couple times until you see how it works. Test the program by storing a 5 in register 0 (5 STO 0) and a 1 in register 3 (1 STO 3). Then set flag 1 (SF 1) and run the program (f D). Try running it with flag 1 clear (CF 1). Try other combinations of values in registers 0 and 3 to see the effect they have on the final display.

Do you see how useful labels, flags and conditional tests can be in helping you create complex programs?

Notes

Indirect Addressing

INDIRECT ADDRESSING

Now that you know something about program loops and labels, it's time to look at another nifty programming tool (ah, the wonders of modern technology)!

When you finish this chapter you will know the basics of indirect addressing. You will finally find out why your calculator has a special I–register, and you will know when to press the [I] key and when to press the [(i)] key. (Basically, you'll know a whole lot.)

WHAT IT IS

Indirect addressing is a tool that can save you lots of time and program memory if you learn to use it properly.

There is more than one way to skin a cat. Likewise, there is always more than one way to write a program. It's possible to use 100 lines of program memory to write a program one way and then turn around and write a program that does exactly the same thing using just 10 lines.

If someone asked you to store a 5 in register 0, you would press the keys 5 STO 0. But there is another way to do this (as always). It's possible to store a 5 in register 0 by pressing the keys 0 STO I; 5 STO (i).

Hmmm..., what's going on here?

Try this: Store PI in registers 0, 1, and 2 without touching the 0, 1, or 2 keys.

Solution:

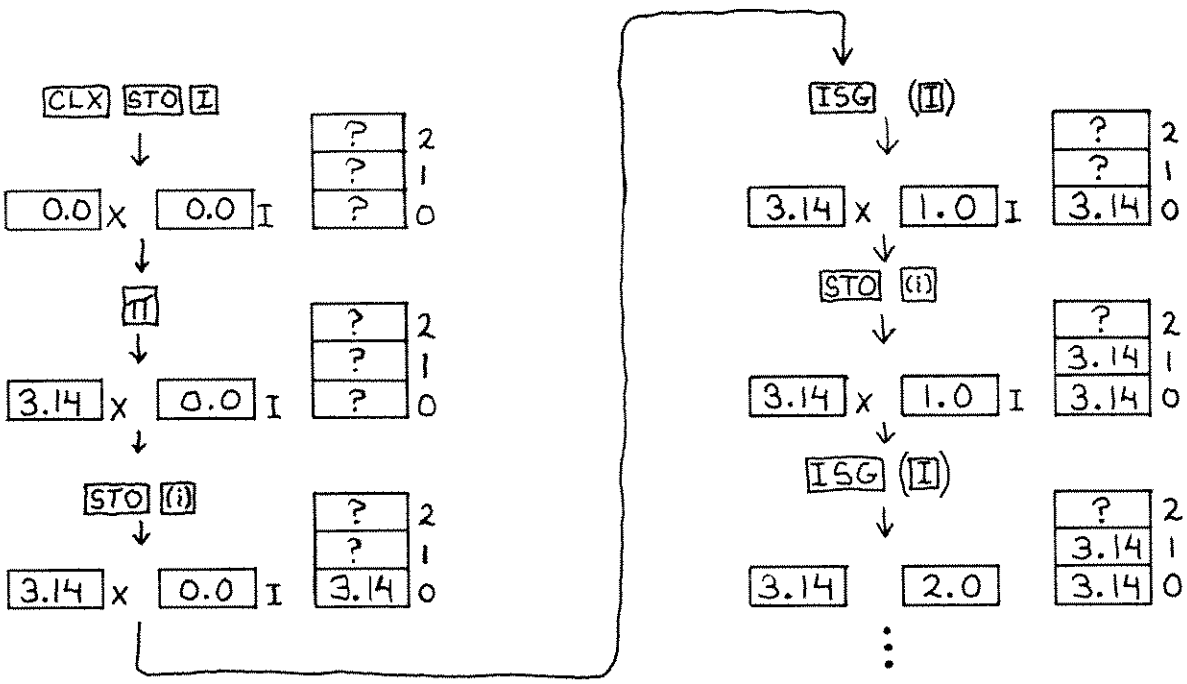
HP–11C	HP–15C
CLX STO I	CLX STO I
f π	g π
STO (i)	STO (i)
ISG	ISG I
STO (i)	STO (i)
ISG	ISG I
STO (i)	STO (i)

(If you understand this solution, go on to page 148.)

Whenever you store a number in a register, you are addressing that register. For example, when you store a number in register 4, you first key in the number, then you press STO 4. By pressing the 4 key you are telling the calculator which register you want to store into. That is, you are addressing register 4. This type of addressing is called "direct addressing."

But this chapter is all about indirect addressing. If you're going to store a number in register 4 using indirect addressing, you need to have a 4 in the I–register; you then key in the number and press STO (i). By pressing the (i) key you are telling the calculator "look in the I–register for the name of the register in which to store." This is indirect addressing.

Watch what happens in registers X, I, 0, 1, and 2 as you key in the solution to the last problem:



And notice when you press STO (i), the calculator looks in the I–register to obtain the name of the register in which to store.

Now clear the X–register (<--); then press RCL (i).

Question: Where did that PI come from?

Answer: Register 2.

There is a 2 in the I–register, so pressing RCL (i) recalls the number from register 2, which is PI.

Try this: Write a program (15 lines or less) that stores the numbers 50 through 59 in registers 5 through 14, respectively.

Solution:

```
001   LBL A
002   5
003   .
004   0
005   1
006   4
007   STO I
008   5
009   0
010   LBL 6
011   STO (i)
012   1
013   +
014   ISG     (ISG I on the HP-15C)
015   GTO 6
```

(If you understand this program, turn to page 152.)

Lines 002 through 009 set up the X– and I–registers like this:

Lines 010 through 015 make up a loop starting with LBL 6.

The first time through this loop, at line 011, the calculator will look at the I–register to find the name of the register in which to store 50 (from the X–register). In the I–register, the calculator finds the number 5.014. But for the address, the calculator looks only at the digits to the left of the decimal point. Thus, this first time through the loop (at line 011), the calculator stores a 50 in register 5. The rest of the loop adds 1 to the 50 in the X–register, increments and tests the I–register (remember how ISG works?), and then goes back to LBL 6.

The second time through the loop, the X– and I–registers look like this:

This time, at line 011, when the calculator looks in the I–register for an address, it will find the number 6.014, so it stores 51 in register 6. Then it adds 1 to the X–register, increments the I–register and tests to see if 7 is greater than 14. Since it isn't, it goes back to LBL 6.

(If you need to review ISG, hold your place here and flip back to page 128 for a quick refresher.)

The calculator will continue around the loop, storing appropriate numbers in appropriate registers, until the 10th time through the loop.

At the beginning of the 10th time through, the X– and I–registers look like this:

The calculator finds 14 (from the 14.014 in the I–register) as the address of the register in which to store that 59 (at line 011). Lines 012 and 013 add 1 to the X–register. Line 014 increments the I–register (making it 15.014) and tests to see if 15 is greater than 14. Since it IS greater, and since the name of the function at line 014 is Increment and SKIP IF GREATER THAN, it WILL skip line 015. That ends the program (whew)!

Try this: Write a program that recalls all those numbers that the previous program stored (in registers 5 through 14) and momentarily displays them in reverse order (i.e. the number in register 14 first).

Solution:

```
001   LBL B
002   1
003   4
004   .
005   0
006   0
007   4
008   STO I
009   LBL 0
010   RCL (i)
011   PSE
012   DSE    (DSE I on the HP-15C)
013   GTO 0
```

(If you're getting the hang of indirect addressing, and you don't need additional explanation of this program, go to page 156.)

Lines 002 through 007 store the control number 14.004 in the I–register (remember control numbers? --page 129). Notice that, as before, this number is used both for loop control (with the DSE function) and as an indirect address (with the RCL (i) function). This will frequently be the case. That is, it's common to use the integer portion of the loop control number as a register address.

Lines 009 through 013 make up a loop starting with LBL 0.

The first time through this loop, at line 010, the calculator will look at the I–register to find the name of the register from which to recall. In the I–register, the calculator finds the number 14.004. So the calculator recalls the value from register 14. Line 012 decrements the value in the I–register by 1, making it 13.004. It then tests to see if 13 is equal to or less than 4 (see page 131 if you're questioning why). Since this is NOT the case, the calculator DOES NOT skip the line immediately following the DSE. Thus, it starts the loop again.

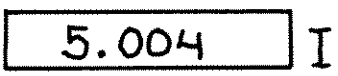
(If all this looping is causing you any slight dizziness, don't worry--this is normal and it will clear up as you go along.)

On the second pass, the I–register looks like this:

This time, when the calculator looks in the I–register for an address at line 010, it will find the number 13.004; so it recalls the number in register 13. Then it decrements the I–register and tests to see if 12 is equal to or less than 4. Since it isn't, it goes back to LBL 0.

The calculator will continue around the loop, recalling numbers from appropriate registers, until the 10th time through the loop.

At the beginning of the 10th time through, the I–register looks like this:



This time the calculator recalls the number in register 5 (line 010), displays it for a moment (line 011); and at line 012 it decrements the I–register making it 4.004. Since 4 is equal to or less than 4, line 013 is skipped, and this ends the program.

(Now, if you have an HP–11C, just go to the top of the next page.)

REGISTERS 20–65 ON THE HP–15C

On the HP–15C, any data register above register .9 can only be addressed indirectly. So, for example, to store the number 87.2 into register 35, you would use these keystrokes: 35 STO I 87.2 STO (i), assuming the top of the data memory has been set somewhere above register 35 (35 f DIM (i)).

Because indirect addressing is mainly a programming tool, and because these registers (20–65) can only be addressed indirectly, you will find that you will rarely use these registers except with programmed instructions.

As an exercise in using these registers in a program, you might want to rewrite the program on page 148 so that it stores the numbers 50 through 59 in registers 15 through 24.

INDIRECTLY ADDRESSING LABELS

Once you understand that you can use the I–register to specify the address of a register, it's good to know that you can indirectly address some other things in your calculator. On the HP–11C and HP–15C, you can indirectly address labels in programs. Also, on the HP–15C (only), you can indirectly address flags.

As you know, you should refer to the capital "I" whenever you want to ALTER the contents of the I–register itself. But whenever you want only to EXAMINE these contents––to obtain the address of another register––you should use the lower–case (i) notation.

Well, just as you are getting that straight, the rules are going to change on you. It turns out that the above rules apply when you are indirectly addressing REGISTERS, but not for other indirect operations. So remember: the only time you will use the [(i)] key is when you are indirectly addressing a register. You don't use this key to indirectly address a label. If you want to go to a label whose address is in the I–register, you press [GTO] [I]. You use the [I] key, not the [(i)] key. We'll show you an example of indirect addressing of labels in the program development chapter that follows.

At this point, take a breather and look back at how far you've come:

You know all about the keyboard, the stack, the display and the data registers, and how to use STO and RCL.

You know how to use the stack for arithmetic and how to use ENTER, CLX and LSTX.

You know about program memory and how to edit a program.

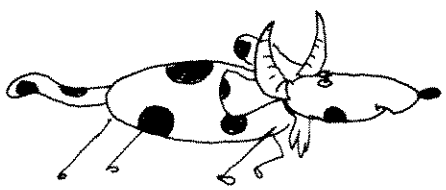
You know how to use labels and GTO and GSB to tell the program pointer how to "jump" around to different points in a program.

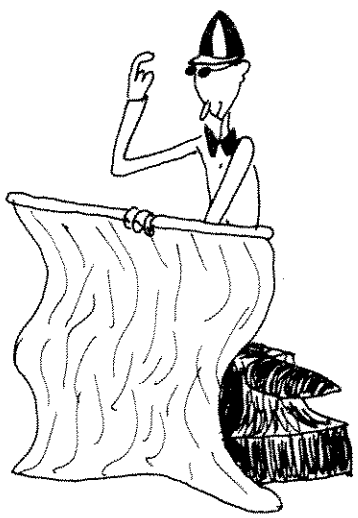
You know how conditional tests work and how to use flags, ISG and DSE to form program loops or make decisions.

You know how to use indirect storage and recall with the data registers.

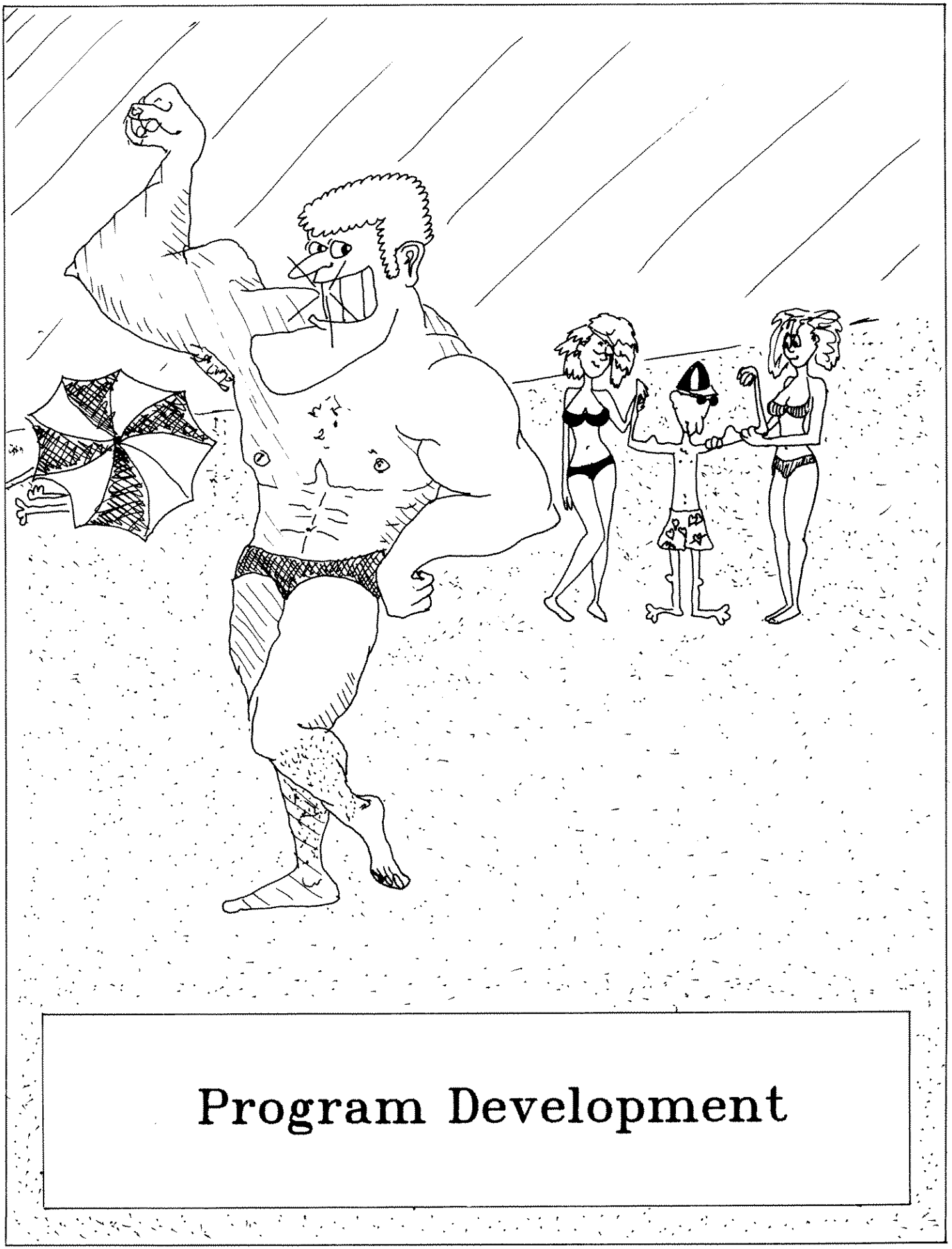
You know that you would probably like some more practice and review of all this--to clear up the fog that may be lingering.

Very well. Step right this way.... ---------->





Notes



Program Development

PROGRAM DEVELOPMENT

Now you are about to embark on the subject that (as the title indicates) is the purpose of this book. This chapter will walk you through the development of three programs. They are fairly simple applications, but each one demonstrates certain methods of programming on the HP–11C and HP–15C.

CHECKBOOK BALANCING

This first program is mainly an exercise to get you warmed up. It is a fairly common application––balancing a checkbook––and you will probably find that the program does not really do much to help with that task. But the idea of designing and writing the program is important.

Checkbook balancing is something most of us have experienced (or at least the ATTEMPT is familiar).

Try this: There are three major steps to balancing your checkbook. Write down these three steps.

Solution:

1. Find the balance from the last time you balanced your checkbook (and hope this wasn't too many months ago).

2. Add all the deposits and interest since that time. Keep a running balance of the result of each addition.

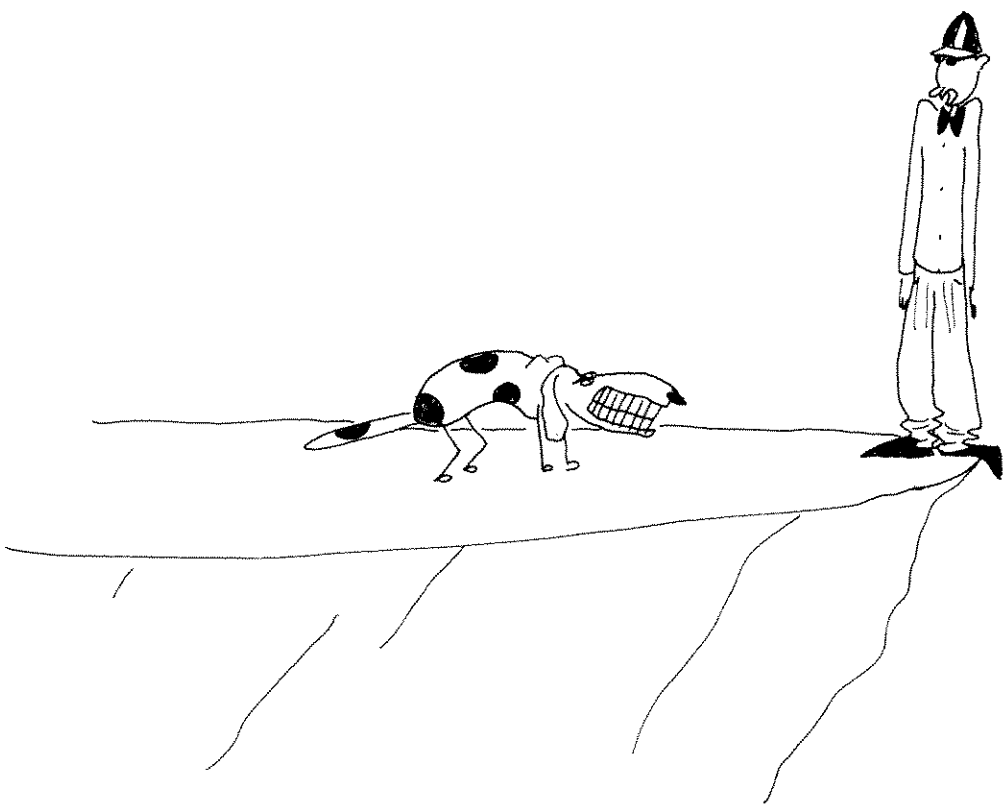
3. Subtract all checks and charges. Keep a running balance of the result of each subtraction.

After you perform the above three steps, your checkbook will be balanced. Basically, what you've done here is defined the process by which you balance your checkbook, in terms that are easy to understand.

You could hand this three-step list to your friends, and they could follow it with little or no problem.

So in a sense, your checkbook-balancing program is complete. You understand the problem, and with these three steps, you've developed a process to handle the problem.

So what's the big deal? (Read on ------>)



Question: What would happen if you were to explain a recipe to a small child (4 to 6 years old) in the following manner?

"Get a cup of flour, 1/3 cup shortening, 1/2 teaspoon salt, and a tablespoon of sugar. Then cut the shortening into a mixture of the flour, salt, and sugar, until the chunks are about the size of peas. Next, add a couple tablespoons of cold water and blend with a fork (not too much). Shape the dough into a ball, and roll it out. That's it! Pie crust!"

Answer: The child would be dumbfounded.

Nevertheless, this is the way we all think. As we develop and gain experience in life, many of the details of day-to-day tasks become automatic; they require no conscious thought. Driving a car is a good example of such details becoming automatic.

But it takes patience and effort to explain even a "simple" process to a small child, or to someone who is unfamiliar with the basic details of that process. So we actually have to slow down our thinking process and analyze each step.

If you were explaining our pie crust recipe to a small child, the first step of the recipe would probably translate from "get a cup of flour" to something like this:

"Now listen, here is a one-cup measure (note the visual aid). Go over to the flour can and scoop out one cup of flour. You'll need to use this knife to level off the top, so you have exactly one cup of flour."

So you see, it takes more words to explain a process to a small child because a child's thinking process is less complex than ours.

Well, to complete the analogy, the "thinking process" of a calculator is far, far less complex than ours. In order to program the HP-11C or HP-15C to do a task, you must first list the steps by which you would handle the task yourself, then expand each of those steps into several, much simpler tasks. In other words, you have to translate your complex thinking process into simple terms that the calculator can understand.

Try this: Rewrite the three checkbook-balancing steps in terms that are closer to the way your calculator "thinks."

Solution:

1. Store initial balance in a numbered register.

2. Wait for an input of either a check, charge, deposit, or interest.

3. If the input is a check or charge, subtract it from the balance.

4. If the input is a deposit or interest, add it to the balance.

5. Display the new balance.

6. Go back to step 2.

Your solution may vary considerably from ours. This is where you start to develop your own programming style. Everyone will approach a solution in a slightly different manner. Try to follow our solutions the first time through these programs; then feel free to experiment with your own solutions.

You can see that what were three general steps have evolved into six detailed steps. And each of these six steps carries a simpler concept than each of the original three. And each of these six steps "sounds closer" to the language of your calculator.

Try this: Equipped with the previous six steps, develop a program to balance your checkbook.

Solution:

```
001   LBL B
002   STO 05
003   R/S
004   LBL C
005   CHS
006   LBL D
007   STO + 5
008   RCL 5
009   R/S
```

(If this makes sense to you, head to page 174.)

If you're even slightly confused, that's good. A completely unexplained list of calculator code should indeed be confusing.

What follows is an explanation of the thought process you might use to proceed from the six steps on page 166 to the 9 lines of code above.

First, don't expect to start at the top of the list. Of course, the order of the steps will reflect the order in which things will eventually be done in the completed program. But that doesn't mean you're going to start DEVELOPING the program at step 1.

The first thing to do is search through the list for the steps that are most significant to the program. Basically, what you're looking for are the steps that look like they will take the most work. You will develop these steps first and then design the rest of the program around them.

In our list of six steps, steps 3 and 4 are the only steps that will require some type of calculation and some type of decisionmaking:

3. If the input is a check or charge, subtract it from the balance.

4. If the input is a deposit or interest, add it to the balance.

Looking at steps 3 and 4, you can see that your calculator has to treat an input in one of two ways, depending on whether it is a check or a deposit. Now it boils down to this: one way or another, you are going to have to tell the calculator whether you are keying in a check or a deposit amount.

There are many ways to tell this to your calculator. One of the easiest ways (we think) is to have one key to press for a check (or other charge) and another key for a deposit (or other credit).

So, when the program is complete, we want to be able to key in an amount, press f C, and let the calculator treat that amount like a check (that is, it would subtract that amount from our balance). Likewise, we want to be able to key in an amount, press the D key, and let the calculator treat that amount like a deposit.

The only difference between the way a check is treated and the way a deposit is treated is that a check is subtracted from the balance, while a deposit is added to the balance. Other than that, the program should treat a check the same as a deposit.

Now look at a little arithmetic. Probably everyone remembers that adding the negative of a number is just like subtracting that number. That is,

$$A - B = A + (-B)$$

So, when we press the [C] key, if the calculator just puts a negative sign on the amount we've keyed in, then treats it like a deposit--thus ADDING this (negative) number to the balance--that should do the trick!

With all this in mind, we can sketch out a routine to handle steps 3 and 4 of our list:

```
LBL C
CHS
LBL D
RCL BALANCE
+
STO NEW BALANCE
```

Of course, there are no "RCL BALANCE" or "STO NEW BALANCE" functions on your calculator. We haven't yet designated a storage register for keeping the balance. But by using this terminology, we can organize our thoughts in a language that is close to what the calculator uses but that we can still understand, too.

These six lines will just about take care of steps 3 and 4 of our list. Now we are going to put those steps on the back burner for awhile, and look at the others instead.

Try this: Translate--into the language of your calculator--steps 1 and 2 ("store initial balance in a numbered register" and "wait for an input").

Solution:
```
LBL B
STO 5
R/S
```

We have chosen the [B] key to indicate "initial balance," and we have chosen register 5 to store that balance. So with this little routine in your program memory, you can key in the initial balance of your checking account, press [f] [B], and the calculator will copy the number in the X-register into register 5.

Try this: Put steps 1, 2, 3, and 4 together, and rewrite steps 3 and 4, now that you know where the balance is stored (register 5).

Solution:

```
LBL B                    LBL B
STO 5                    STO 5
R/S                      R/S
LBL C        or          LBL C
CHS                      CHS
LBL D                    LBL D
RCL 5                    STO + 5
+
STO 5
```

Either of the above routines will take care of steps 1 through 4. The routine on the left will even take care of steps 5 and 6. But the routine on the right will not do step 5 of our list unless we add RCL 5 at the end. Even then, the routine on the right will be the shorter of the two, so let's look closely at how it works:

```
001   LBL B
002   STO 5
003   R/S
004   LBL C
005   CHS
006   LBL D
007   STO + 5
008   RCL 5
```

First, key this program into your calculator. Starting in RUN mode, here are the keystrokes:

KEYSTROKES	DISPLAY
g P/R	
f CLEAR PRGM	000–
f LBL B	001– 42, 21, 12
STO 5	002– 44 5
R/S	003– 31
LBL C	004– 42, 21, 13
CHS	005– 16
LBL D	006– 42, 21, 14
STO + 5	007– 44, 40, 5
RCL 5	008– 45 5
g P/R	RUN mode display

Now the program is ready to use. It is designed so that when we want to balance our checkbook, all we have to do is key in the balance from the last time we balanced it, and press f B (for Balance).

We then key in the amounts of any checks we have written or deposits we have made and press f C (for Checks) or f D (for Deposits). Simple!

Try this: The last time you balanced your checkbook your balance was 1422.56. Since then, you have written 3 checks (27.22, 96.00, and 445.75); you also deposited your weekly paycheck (1242.32) and a dividends check from an investment you made several years ago (377.85). Using the checkbook balancing program you just keyed in, find the current balance of your checkbook.

Solution: 2473.76

(If you need no more explanation of this program, go to page 177. But if you're in the slightest doubt....--->)

First, you need to key in the most recent balance of 1422.56 and press f B. The calculator will flash "running" at you for a split second, and then it will stop with 1422.56 in the display. This balance is now stored in register 5.

Then you simply key in 27.22 f C for the first check. The calculator will subtract 27.22 from the balance in register 5 and display the new balance. Then you have to key in the next two check amounts, pressing f C after each one.

Finally you key in the deposits, one at a time, pressing f D after each one. Each deposit will be added to the balance in register 5 and the new balance will be displayed. The final result is 2473.76. (Let's hope you have a checking account that earns interest!)

As you'll recall, our program looks like this:

```
001   LBL B
002   STO 5
003   R/S
004   LBL C
005   CHS
006   LBL D
007   STO + 5
008   RCL 5
```

Is there anything we can add to this program to make it better? Hmmm...let's see....

How about this? Since we're always dealing in dollars and cents when we run this program, let's make the program set the display to show only two decimal places (FIX 2).

Try this: Insert a line in the program so that when you key in the initial balance, the program will set the display to show only two decimal places.

Solution:

```
001   LBL B
002   STO 5
003   FIX 2
004   R/S
005   LBL C
006   CHS
007   LBL D
008   STO + 5
009   RCL 5
```

The line FIX 2 is inserted right after line 002 in the program. (If you had any difficulty inserting this line, turn to page 109 for a review of program editing. Then come back and continue from here.)

So the checkbook balancing program is complete. You may want to take a break now, make yourself a soothing cup of tea, and use this new program to balance your checkbook.

You'll probably find, when using this program, that it's just about as easy (or even easier) to balance your checkbook by using the stack to do the arithmetic. But developing this program was a good exercise, don't you think?

Notes

FEET, INCHES, AND SIXTEENTHS

"For our next number," we will develop a program to convert feet, inches, and sixteenths of inches into feet and decimal fractions of feet, and vice versa. For example, 1 foot and 6 3/16 inches will be converted to 1.515625 feet; and 1.515625 feet will be converted to 1 foot 6 3/16 inches.

The calculations involved in this program will be relatively simple. But the main emphasis of this program will be the format of the input and output. To put it bluntly, we have to develop a convenient way to input feet, inches, and sixteenths of inches!

Now, it would be best if we could key in one number to represent all three units (feet, inches, sixteenths). To do that, we need to develop a format to represent three different things with one number....

Try this: Convert 3 hours, 26 minutes, and 14 seconds into hours and decimal fractions of hours.

Solution: 3.2614 [g] [->H] (displays 3.43722)

So 3 hours, 26 minutes, and 14 seconds is equal to 3.43722 hours.

The point here is that for certain functions on your calculator, one number can represent different things. The [->H] (convert to Hours) function looks at the number in the X–register and sees the digits as representing Hours, Minutes, and Seconds, in the form HH.MMSS.

The HH means "number of hours," the MM means "number of minutes," and the SS means "number of seconds." So 03.2614 means 3 hours, 26 minutes, and 14 seconds. There are only two places reserved for minutes, because the number of minutes will never exceed 60 (that's an hour), and fractions of minutes are expressed in seconds. (However, there are actually more than two places reserved for seconds. If you want to key in 1 minute, 57 3/4 seconds, you would key in .015775, because 3/4 = 0.75. The fraction of a second is keyed in after the whole seconds.)

Well, why not take this format (that HP has developed) for representing hours, minutes, and seconds--with one number--and adapt it to fit our feet, inches, and sixteenths problem? (Aha!)

Try this: Express 4 feet, 8 and 9/16 inches in a format similar to the HH.MMSS format.

Solution: 4.0809 (FF.IISS)

(At this point, we are, of course, feeling enormously pleased with ourselves for such cleverness.)

This is the input format that we'll use in our conversion program. We will key in feet, inches, and sixteenths of inches, using the form FF.IISS!

Here FF means "number of feet," II means "number of inches," and SS means "number of sixteenths." Two places are reserved for the inches and the sixteenths. This makes good sense because there will be, at most, 11 and 15/16 inches (.1115) in any fraction of a foot.

Try this: Sketch down the general steps required to convert an input of the form FF.IISS (feet, inches, sixteenths), to feet and a decimal fraction of a foot (we'll represent feet and decimal fraction by the normal numerical FF.ffffff...).

Solution:

1. Get the input of the form FF.IISS

2. Take the integer portion (FF) and save it. This is the number of whole feet (it won't change in the final answer). Also, save the fractional portion (.IISS).

3. Multiply this fractional portion by 100 to get (II.SS).

4. The integer portion of (II.SS) represents the number of whole inches, and the fractional portion represents the number of sixteenths. Separate these two portions.

5. Divide the fractional part by 1.92 (.SS/1.92).

6. Divide the number of inches by 12 (to get II/12), and add up all the parts.

7. The result will be the sum: FF + II/12 + SS/192.

What we're shooting for in the solution is this:

First, we want to save the FF part of the input because this is the number of whole feet, and we don't need to change this in the final answer.

Next, we need to find the number of inches (II) and divide this by 12 (there are 12 inches in a foot). Then we have to find the number of sixteenths and divide it by 192 (12 × 16), since there are 192 sixteenths–of–an–inch in one foot.

Try this: Sketch down the steps required to take an input of feet and decimal fraction of a foot (FF.ffff) and convert it to feet, inches, and sixteenths (FF.IISS).

Solution:

1. Get an input of FF.ffff.

2. Save the integer portion, FF.

3. Multiply the fractional portion by 12 and save the integer portion of the result, II.

4. Multiply the remaining fraction by 16, SS. (Let's leave fractions of sixteenths just like fractions of seconds.)

5. Save the result in the X–register in the form FF.IISS (FF + II/100 + SS/10,000).

Let's run an arbitrary number through these steps to see if they work. Try, for example, 14.9 feet.

Step 2 says to save the integer portion. The integer portion of 14.9 is 14. So, save 14.

Step 3 says to multiply the fractional portion by 12 to get the inches (12 ✗ .9 = 10.8). Again, we save the integer portion.

So far we have 14 feet, 10 inches, and 8 tenths of an inch. To convert the 8 tenths of an inch into whole sixteenths, multiply the .8 by 16. This gives 12.80, so the final result is 14 feet, 10 and 12.80/16 inches.

Our general idea seems to be working!

From the way things are working out, it looks like we are going to develop two independent routines. One of these routines will take an input of the form FF.IISS and return feet and decimal fractions of feet (FF.ffff). The other program will take an input in the form FF.ffff and return FF.IISS. These two routines will be a matched pair of functions much like the [->H] (convert to hours) and [->H.MS] (convert to Hours Minutes and Seconds) functions.

If we combine our lists and put them under labels A and B, they might look like this:

1. LBL A (input: FF.IISS).
2. Save the integer portion of the input: FF.
3. Multiply the fractional portion by 100, to get II.SS.
4. Save the integer portion: II.
5. Divide the fractional portion by 1.92, to get SS/192.
6. Divide the inches by 12, to get II/12.
7. Result = FF + II/12 + SS/192.
8. END of LBL A (output: FF.ffff).
9. LBL B (input: FF.ffff)
10. Save the whole feet: FF.
11. Multiply the fractional portion by 12, to get II.
12. Multiply the remaining fraction by 16, to get SS.
13. Result = FF + II/100 + SS/10,000.
14. END of LBL B (output: FF.IISS).

Try this: At this point, you should be able to take the list of sixteen steps from the previous page and convert them to HP–11 or HP–15 program lines. Give it a try and see if you can come up with a workable solution. It's possible to do all the calculations using only the stack registers.

Solution:

001	LBL A	012	.
002	INT	013	9
003	LSTX	014	2
004	FRAC	015	÷
005	EEX	016	X<>Y
006	2	017	1
007	x	018	2
008	INT	019	÷
009	LSTX	020	+
010	FRAC	021	+
011	1	022	RTN

023	LBL B	034	6
024	INT	035	x
025	LSTX	036	EEX
026	FRAC	037	2
027	1	038	÷
028	2	039	+
029	x	040	EEX
030	INT	041	2
031	LSTX	042	÷
032	FRAC	043	+
033	1	044	RTN

If you have no problems keying in the above program, and you're sure you understand how it has been developed then cruise on ahead to page 199. Otherwise... ------>

Let's take a quick (exciting) look at how each step in the list on page 190 transforms into this final listing of program code.

1. LBL A (input FF.IISS):

   001 LBL A

   The first step translates directly over. When you press f A to execute the complete program, the calculator will assume that an input in the form of FF.IISS is sitting in the X–register.

2. Save the integer portion of the input:

   002 INT

   The INT function saves, in the X–register, the integer portion of the input, which is FF. The original input, FF.IISS, is saved in the LASTX register.

3. Multiply the fractional portion by 100:

```
003   LSTX          006   2
004   FRAC          007   x
005   EEX
```

Before we multiply the fractional portion by 100, we have to get the fractional portion into the X–register. The original input is in the LSTX–register (Do you remember why? See page 56). Line 003 recalls the original input; line 004 takes the fractional portion of it; lines 005, 006, and 007 multiply the fractional portion by two.

4. Save the integer portion of this result:

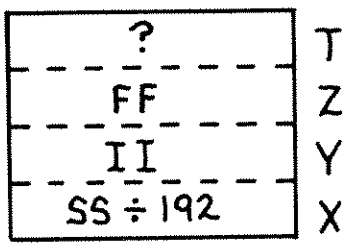
```
008   INT
```

This saves II in the X–register (and II.SS in the LASTX–register).

5. Divide the fractional portion of II.SS by 1.92:

```
009   LSTX          013   9
010   FRAC          014   2
011   1             015   ÷
012   .
```

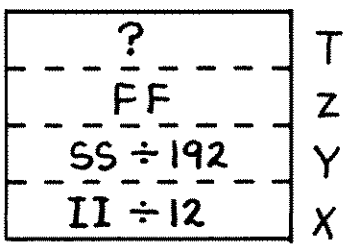
Dividing .SS by 1.92 gives the same result as dividing SS by 192. Right? Now the stack would look like this:



6. Divide the inches by 12:

```
016   X<>Y          018   2
017   1             019   ÷
```

The inches (in the Y–register) are brought into the X–register and divided by 12. The stack looks like this:



7. Result (FF.ffff...) = FF + II/12 + SS/192:

   020 +
   021 +

   The final result ends up in the X–register.

8. END of LBL A:

   022 RTN (the output is FF.ffff)

9. LBL B (input: FF.ffff):

   023 LBL B

   This is the beginning of the routine that converts an input of feet and decimal fraction to feet, inches, and sixteenths. The input of FF.ffff is assumed to be in the X–register.

10. Save the whole feet:

    024 INT

    The whole feet are saved in the X–register (and the original input is saved in LASTX).

11. Multiply the fractional portion by 12:

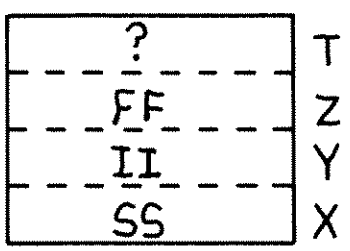
```
025  LSTX        028  2
026  FRAC        029  x
027  1           030  INT
```

This brings the original input back into the X–register and multiplies the fractional portion (.ffff) by 12. The last statement (INT) saves the integer portion in the X–register.

12. Multiply the remaining fraction by 16:

```
031  LSTX        034  6
032  FRAC        035  x
033  1
```

The result of multiplying .ffff by 12 is in the LASTX–register, so we need to call it up and get the fractional portion. Then we multiply it by 16. Now all the numbers have been calculated. From here, it's just a matter of moving each part into its proper decimal place in the output. The stack looks like this:



13. Result = FF + II/100 + SS/10,000:

036	EEX	040	EEX
037	2	041	2
038	÷	042	÷
039	+	043	+

In this final segment program we first rearrange the above equation to this: Result = FF + (II + SS/100)/100.

In this form the equation is easier to translate into program code. With SS in the X–register, all the program does is divide by 100 (Remember EEX from page 27?), add that result to II (in the Y–register), divide again by 100, and add that to FF.

14. END of LBL B: 044 RTN (output: FF.IISS)

Try this: Convert 4 feet, 11 and 5/16 inches to feet and decimal fraction.

Solution: Assuming you have the program keyed in (from pages 191–192), you key in 4.1105 f A. The answer is 4.9427 feet.

Try this: Convert 7.0729 feet to feet, inches, and sixteenths.

Solution: Key in 7.0729 f B to get 7.0014

The conversion routine is complete!

Do you see how much work you can make the calculator do for you?

Are you starting to get the knack of manipulating numbers through program steps?

Just to make sure, here's one more good practice example. ------>

GRAPHING AN EQUATION

The final program we will develop is a program to evaluate a table of X– and Y–values for an equation that you program into the machine. How does that sound? (It really isn't very difficult.)

Most of us have done this before in an old algebra class (long, long ago, in a galaxy far, far away...). Back in algebra, they called it "graphing" an equation or "plotting" an equation. We might start, for example, with some equation like $Y = X^2 + 5X$.

Now remember, these Y's and X's don't have anything to do with the X– and Y–registers in the stack. We could easily rewrite this equation as: $B = A^2 + 5A$, or $(OUTPUT) = (INPUT)^2 + 5(INPUT)$.

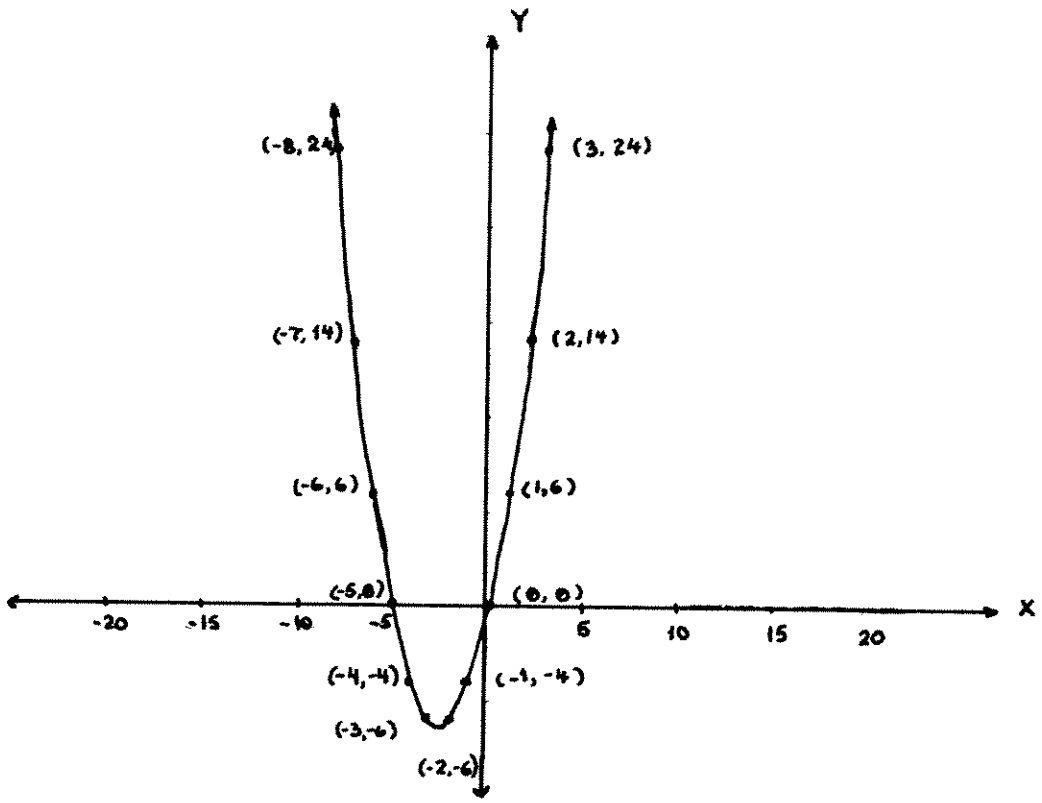
But, because it's common to talk about equations graphed on the X and Y axis, we'll use the original $Y = X^2 + 5X$. So to keep our thoughts clear, whenever we mean the X–register, we'll write "the X–register." Otherwise, X means the variable number X in the equation, OK?

Anyway, to plot the equation $Y = X^2 + 5X$, we would start at, say, zero and plug in a bunch of values for X--to see what Y-value each X gives.

We would generate a table that looks like this:

For X equal to	Y equals
0	0
1	6
2	14
3	24
-1	-4
-2	-6
-3	-6
-4	-4
-5	0
-6	6
-7	14
-8	24

Then we could take these pairs and plot them on a graph with the X–value plotted horizontally and the Y–value plotted vertically:



The program we develop will generate the X,Y pairs, so that we can plot the graph of any equation we program into our HP–11C or HP–15C.

Try this: Write a routine to do the equation:

$$Y = 4X^3 - 12X + 5$$

or

$$\text{OUTPUT} = 4 \times (\text{INPUT})^3 - 12 \times (\text{INPUT}) + 5.$$

Solution:

	001	LBL 0
	002	STO 5
	003	3
	004	Y^x
	005	4
	006	x
	007	RCL 5
	008	1
	009	2
	010	x
	011	–
	012	5
	013	+
	014	RTN

(If translating an equation into a program has you flustered, then take a break and go back to pages 64–93. It will refresh your memory; then you can come back and start again at the top of this page.)

Try this: Use the above routine to create a table of X,Y pairs for the equation $Y = 4X^3 - 12X + 5$. Start with X = 0 and run the program. The result is Y. Then increase the X–value by 0.25 each time, up to 2. You will generate 9 pairs of X and Y.

Solution:

For X equal to	Y equals
0	5.0000
0.25	2.0625
0.50	–0.5000
0.75	–2.3125
1.00	–3.0000
1.25	–2.1875
1.50	0.5000
1.75	5.4375
2.00	13.0000

Got it? (Partly?)

All it amounts to is this: To get the first Y–value (after you've keyed in the program), key in 0 GSB 0, and your calculator will return 5.0000 (if you're set to FIX 4). So the first X,Y pair is 0,5. An X–value of zero gives a Y–value of 5. Plain and simple.

To get the next Y–value, key in .25 GSB 0, and your calculator will return 2.0625. Keep going until you have nine X,Y pairs.

Try this: Write out, step by step, what you just did on the last page, as if you were explaining the process to a friend.

Solution: "Well, let's see...."

1. "I read the problem and saw that I was supposed to use the LBL 0 routine to generate a table of X,Y pairs. I was to begin at X=0, end at X=2, and increase by 0.25 each time. So I was told the name of the program with which to evaluate the Y-values, the beginning X-value, the ending X-value, and the increment.

2. "I started at the beginning X-value.

3. "I ran this current X-value through the LBL 0 routine to get Y.

4. "I wrote down the X,Y pair.

5. "I added the increment (0.25) to X.

6. "I checked to see if this new current X-value was greater than 2 (the ending X-value). If it wasn't, I repeated steps 3 through 6. If it was, I stopped, and I smiled."

Question: What are the six steps that a program would perform to do what you did on page 206 (one page back)?

Answer:

1. The calculator needs to know the name of the function (LBL 0) with which to generate X,Y pairs. It also needs to know the beginning X–value (0), the ending X–value (2), and the increment value, (0.25). The calculator should store all this.

2. Start at the beginning X–value.

3. Run this current X–value through the named program (LBL 0) to get Y.

4. Display the current X,Y pair.

5. Add the increment to the current X–value to get a new current X–value.

6. Check to see if this new current X–value is greater than the ending X–value. If it isn't, repeat steps 3 through 6. If it is, stop.

See what we're getting at? These six steps are nearly the same as those on the page 206.

The general steps that YOU take to complete a process are quite similar to the steps a program has to take to complete the same process.

Now it's just a matter of expanding these general steps into calculator program lines.

Let's agree to store the required data as follows:

I–register––Name of the function to plot

register 1––Beginning X–value

register 2––Ending X–value

register 3––Increment value

Now let's look at step 1 in our list of six steps.

1. "The calculator needs to know the name of the function (LBL 0) with which to generate X,Y pairs, the beginning X–value (0), the ending X–value (2), and the increment value (0.25). The calculator should store all this."

Question: How can we handle this step in a program?

Answer: There are many ways to do this. The final goal to every approach is to get the four values into the four registers (the name of a numeric label into the I–register, the beginning X–value into register 01, the ending X–value into register 02, and the increment into register 03.

The way we're going to approach it is this: These four values will be keyed into the stack (X,Y,Z, and T) right before we run our plotting program.

So the first part of the program is going to assume that the T–register contains the function name; the Z–register contains the beginning X–value; the Y–register contains the ending X–value; and the X–register contains the increment, OK?

With this assumption, we can sketch out a routine that could handle step 1 of the 6 step list:

```
LBL A
STO 3  (stores the increment in register 3)
R↑
STO I  (stores––in I––the name of the function to plot)
R↑
STO 1  (stores the beginning X–value in register 1)
R↑
STO 2  (stores the end X–value in register 2)
```

Now, look at step 2:

2. "Start at the beginning X–value."

This step merely reminds us that, on the first time through, the beginning X–value will be the current X–value. The current X–value will always be maintained in register 1.

Try this: Expand the next step for your calculator.

3. "Run this current X–value through the named program (LBL 0) to get Y."

Solution:

```
LBL 1
RCL 1  (recalls the current X–value to the X–register)
GSB I  (branches to the LBL named in the I–register)
```

(No sweat? Then head to page 214.)

Look at it this way: When you were generating the table of X,Y pairs manually, you established a current X–value in the X–register, and then you ran the program under LBL 0. Well, the program you write to generate these X,Y pairs is going to have to go through the same process.

So the line "RCL 1" brings the current X–value into the X–register.

The line "GSB I" says "look at the program name in register I and GSB (Go SuB) that program." This is another form of indirect addressing. You are indirectly addressing a label.

Notice that the keystrokes are [g] [GSB] [I], NOT [g] [GSB] [(i)]. For some reason, HP chose to use the [I] key for indirectly addressing LBL's. In fact, as long as you remember that the [(i)] key is used only for the indirect addressing of REGISTERS, you'll get along fine.

Finally, notice that we put the LBL 1 at the top. Step 6 mentions going back to step 3, so we need a marker (LBL 1) to tell the pointer where to "jump."

Try this: Expand step 4 ("Display the current X,Y pair").

Solution:

```
RCL 1
R/S
```

This will cause the program execution to stop with the current X–value in the display (and the X–register, of course) and the current Y–value in the Y–register. You'll need to write down each X,Y pair so you can plot the equation. So just write down X, press [X<>Y] and write down Y.

To get the next X,Y pair, start the program running again by pressing [R/S].

Now expand the next step:

5. "Add the increment to the current X–value to get a new current X–value."

Solution:

```
RCL 3  (increment)
RCL 1  (current X–value)
+
STO 1  (new X–value)
```

Finally, write the program lines for this step:

6. "Check to see if this new current X–value is greater than the ending X–value. If it isn't, repeat steps 3 through 6. If it is, stop."

Solution:

```
RCL 2
RCL 1
X≤Y
GTO 1
RTN
```

Now put it all together(!)

```
001   LBL A
002   STO 3  (stores the increment in register 3)
003   R↑
004   STO I  (stores––in I––the name of the function)
005   R↑
006   STO 1  (stores the beginning X–value in register 1)
007   R↑
008   STO 2  (stores the end X–value in register 2)
009   LBL 1  (beginning of loop)
010   RCL 1  (recalls the current X–value)
011   GSB I  (branches to the LBL named in the I–register)
```

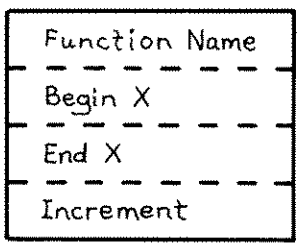
```
012   RCL 1
013   R/S
014   RCL 3  (increment)
015   RCL 1  (current X-value)
016   +
017   STO 1  (new X-value)
018   RCL 2
019   RCL 1
020   X≤Y
021   GTO 1  (end of loop)
022   RTN
```

Key this program into your machine, but DON'T ERASE the program that is there under LBL 0. Make sure your calculator is in RUN mode and press [g] [RTN]. Then put it into PROGRAM mode ([g] [P/R]), and the display will show 000- . Now, as you key in the above program steps, the other program steps will be pushed farther down into program memory.

Before you can use this plotting program, you have to do two things:

1. You need to key in a program under some numeric LBL other than LBL 1. This program will represent the function you want to plot.

2. You need to set the stack up like this:



Try this: You have two programs in your calculator—the plotting program and the program under LBL 0 that gives Y values for the equation: $Y = 4X^3 - 12X + 5$ when you input X.

Now generate a table of X,Y pairs for this equation beginning at −2.2, ending at 0.4, and increasing by 0.2 each time.

Solution: 0 [ENTER] 2.2 [CHS] [ENTER] .4 [ENTER] .2 [f] [A]. The calculator will stop with the first X−value in the X−register and the first Y−value in the Y−register. Write down the first X−value, press [X<>Y], and write down the first Y−value. Then press [R/S] to compute the next X,Y pair etc.

For X equal to	Y equals
−2.2	−11.1920
−2.0	−3.0000
−1.8	3.2720
−1.6	7.8160
−1.4	10.8240
−1.2	12.4880
−1.0	13.0000
−0.8	12.5520
−0.6	11.3360
−0.4	9.5440
−0.2	7.3680
0.0	5.0000
0.2	2.6320
0.4	0.4560

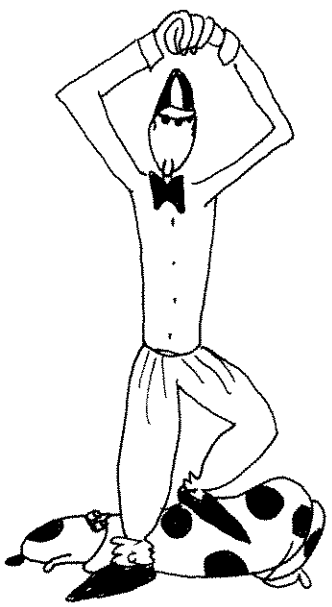
It's easy!

This is what programming is all about. You must think carefully just once--when you write the program--and then you can make your calculator do all of the boring, repetitive work for you--over and over!

But remember! You have to SPEND some time first in order to SAVE time later. So keep practicing--with these programs or with others of your own choosing.

And for other helpful ideas for getting the most out of your calculator, you might want to read through the appendices that follow. But as of now, you have graduated from the Easy Course.

CONGRATULATIONS!



THE END

Appendices

APPENDIX 1
Using The Manuals

In this course, we gave you some visualization tools (ways to picture your calculator's insides) and a fundamental knowledge of how your calculator works. How much you retain will depend upon your calculating needs and how much you practice with your calculator.

Also, once you have this basic knowledge of the operation of your calculator, it is much easier to work with the manuals that came with your calculator and with other HP books. So at this point, the manuals are good sources to use in expanding your knowledge of the works of your machine. You will find them to be excellent books for this kind of continued reference.

To use the manuals effectively for reference, you need to learn how to use the indexing in the back of those manuals. There is a "Function Summary and Index" or "Function Key Index," in case you need to know more about using a particular function on your calculator; and there is a "Subject Index," where you can look up more general information about a particular subject.

In order to look up something quickly, you have to know enough about your calculator to have in mind a word or two that you can refer to as a possible lead. But it's a little bit like the old dictionary dilemma: "How do you look up the spelling of a word in a dictionary if you don't know how to spell it?"

Well, the best way to familiarize yourself with your manual and its index is to skim through the entire book, working examples here and there; also, skim through the indexes. After all, let's face it: Reading the whole manual thoroughly would take the better part of a long time. And few people need to know EVERYTHING about their calculator. Some people may use the hyperbolic functions on their calculator every day, and some people may never use them. The same goes for statistical functions, etc. It's all up to you and how you use your calculator.

This course has given you the basic vocabulary you need to look up those subjects you want to study more thoroughly. Hopefully, with this vocabulary, you can now ask yourself questions such as these:

"Do I really know the effect that the [%] FUNCTION has on the STACK? It's not acting as I think it should, so maybe I should look it up in the FUNCTION INDEX to learn the details."

or

"Is something weird happening in the DISPLAY? Maybe I'll find the answer by looking up DISPLAY in the Subject Index."

or

"Do I really know all the details of INDIRECT ADDRESSING?"

or

"Do I need to review STORING NUMBERS?"

If you can ask yourself questions like these whenever you have a problem, you'll be able to find solutions by using your manuals for reference.

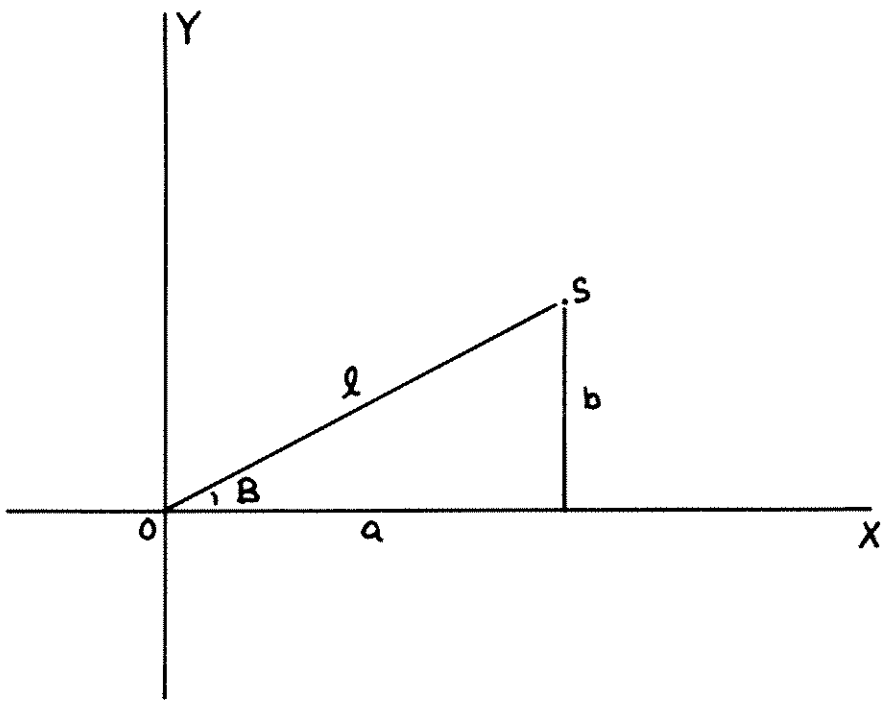
And occasionally, you may even see an ERROR message in the display. If you do, and if you have no idea why, just remember the last function you executed (or put your calculator into program mode to see the program line that caused the ERROR), and then look up the ERROR number in Appendix A of your manual.

Appendix 2

Trigonometry and Vectors with →R and →P

The names of the functions, →R ("convert to Rectangular coordinates") and →P ("convert to Polar coordinates"), may not fully reveal the usefulness of these functions. Anyone who solves trigonometry problems or works with vectors (in science and engineering) should be using these functions regularly.

The easiest way to describe what these functions do is to look at a right triangle placed on an X,Y plane:



The names of these functions (→R and →P) come from the fact that there are two ways of describing the location of point S.

Using rectangular coordinates, you would say "from the origin (point 0) move the distance 'a' in the X direction, then move the distance 'b' in the Y direction to get to point S."

Using polar coordinates, you would say "from the origin (point 0) move the distance 'l' at an angle B from the X–axis to get to point S."

So the names come from describing points in two–dimensional (X,Y) space. To describe a point, you can either specify an X–distance and a Y–distance, or you can specify one distance and an angle. OK?

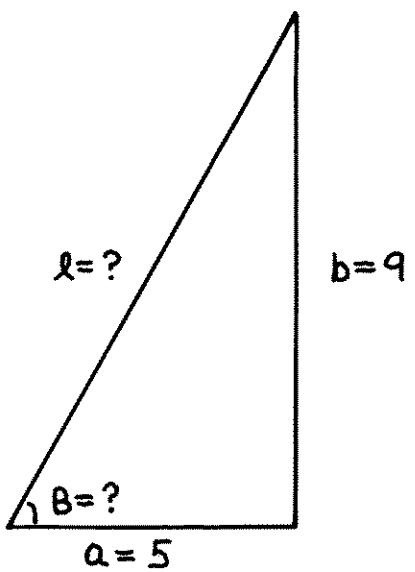
But describing points is not the most exciting thing in the world. Few people can make a living by describing points, no matter how many different ways they can describe them.

So it's important to notice that we are not just describing a point. We are describing a right triangle and also a vector.

To describe a right triangle, you need only specify the length of the legs (like 'a' and 'b') or you can specify the length of the hypotenuse ('l') and one internal angle ('B'). Either way, you would completely describe the triangle.

To describe a vector, you can either specify its X and Y components or you can specify its magnitude and direction on an X,Y plane.

Try this: Calculate the length of the hypotenuse of this triangle and the angle B (in degrees).

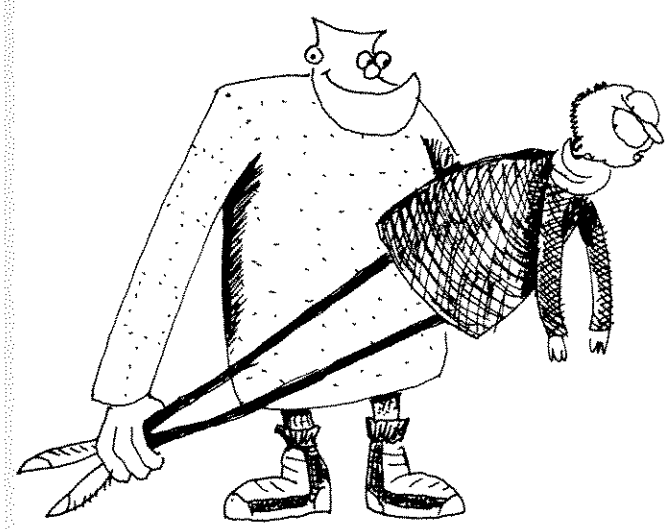


Solution: 9 ENTER 5 g →P

The length of the hypotenuse is now in the X–register (10.30) and the angle B is in the Y–register (assuming your calculator is in degrees mode, press [X<>Y] to see 60.95 degrees).

Press [X<>Y] to get the angle back into the Y–register. Now convert back to a rectangular description by pressing [f] [->R]. Now 5 is back in the X–register and 9 is in the Y–register!

It's easy to convert back and forth from the "polar description" of a right triangle to its "rectangular description." Just remember that, when converting to the "polar description," the orientation of the right triangle will determine which angle is computed. If, in the above problem, you put a 5 into the Y–register and a 9 into the X–register, the hypotenuse will still be the same, but the calculator will return the complementary angle.



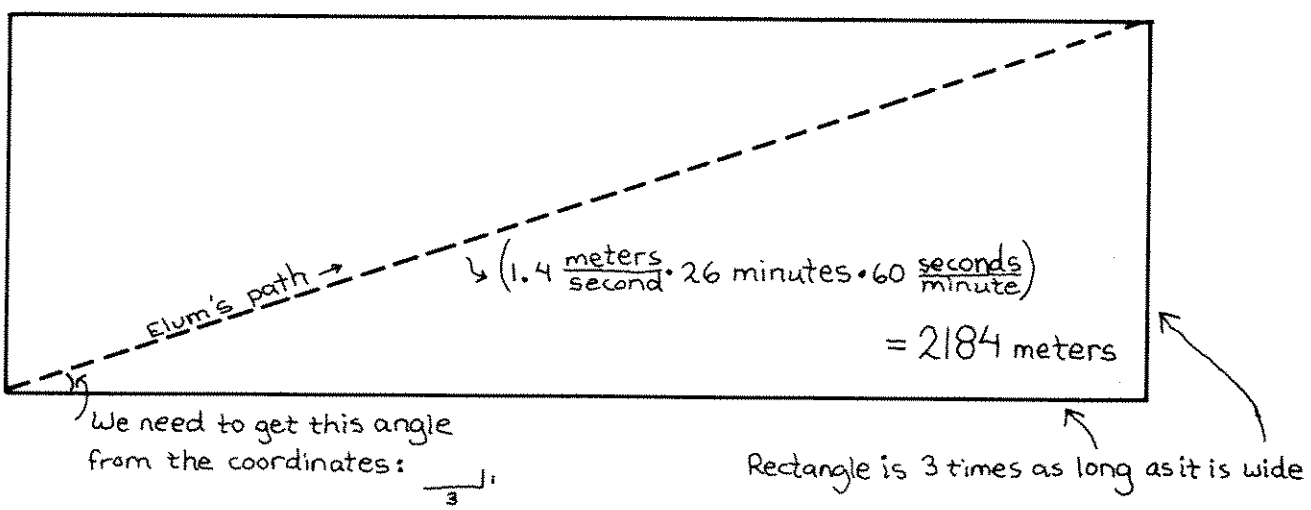
Solve this: Farmer Smith has a mule named Elum that has a consistent walking pace of exactly 1.4 meters/second. The other day Farmer Smith timed Elum crossing one of his rectangular fields diagonally and it took Elum 26 minutes. Farmer Smith knows from windrowing hay in this field that it is three times as long as it is wide.

If Farmer Smith wants to fence in that rectangular field for pasture, how many meters of fencing material should he buy?

Solution: 1 ENTER 3 g ->P X<>Y
1.4 ENTER 26 X 60 X
f ->R + 2 X

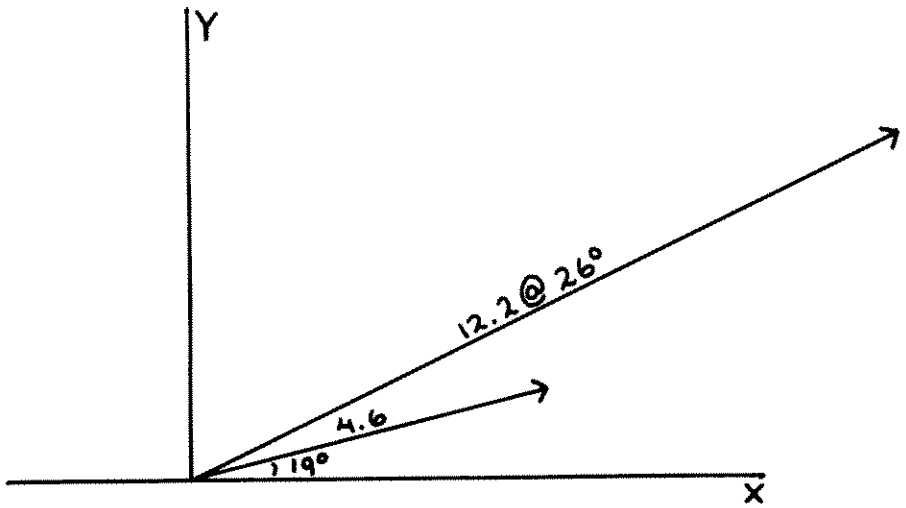
Answer: 5,525.13 meters

Here's how we did this:



SUMMING VECTORS

Try this: What is the resultant of summing these vectors?



Solution: 19 ENTER 4.6 f →R
STO 0 X<>Y STO 1
26 ENTER 12.2 f →R
STO + 0 X<>Y STO + 1
RCL 1 RCL 0 g →P

Answer: 16.78 @ 24.08deg

To sum vectors, you first need to break them into their X and Y components. This is done using the →R function. Then, you simply sum the X components, then the Y components, to get the X and Y components of the resultant vector. Finally, you can convert back to a magnitude and angle using the →P function. This is what we did here.

Here's a program you can use to sum vectors. With this program in your machine, simply key in the angle of the first vector, press ENTER, key in the magnitude of the first vector, ENTER, key in the angle of the second vector, ENTER, key in the magnitude of the second vector, and press f A. The resultant vector's magnitude ends up in the X–register and the angle ends up in the Y–register.

```
001   LBL A
002   ->R
003   X<>Y
004   R↑
005   R↑
006   ->R
007   R↑
008   +
009   R↓
010   +
011   R↑
012   ->P
```

To solve the previous problem––using this program––the keystrokes would be:

19 ENTER 4.6 ENTER 26 ENTER 12.2 f A

The answer is the same, with the magnitude appearing in the X–register and the angle in the Y–register.

Appendix 3
(Features of the HP–15C)

FLAGS

The HP–15C has 10 flags you can use in your programming (flags 0 through 9). As you know, a flag is like an indicator switch you can turn on or off, or you can simply check its status without changing it. By setting a flag (using SF), you are switching it on, and by clearing a flag (using CF), you are switching it off. This feature is useful for making decisions in programming (if you are having trouble using flags, you may want to review pages 123–127 in this book).

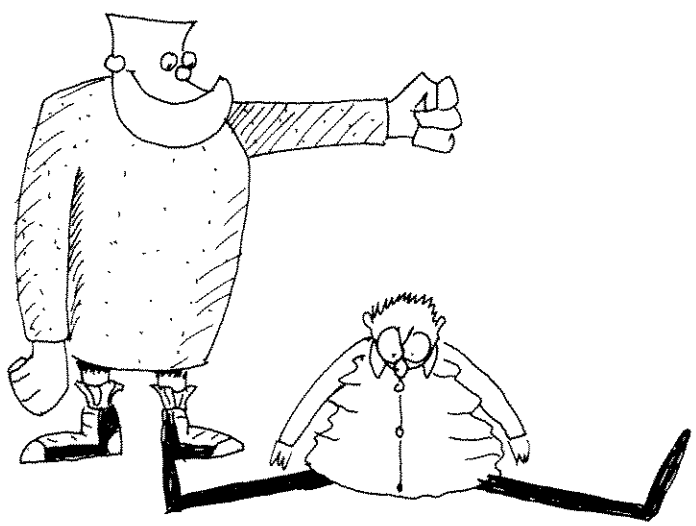
All of the flags except flag 9 will stay in the positions to which you set them until you change those positions (or a program changes them).

One more thing: Flags 8 and 9 have special meanings to the calculator. Flag 8 turns on a little 'C' in the display to indicate that the calculator is in Complex mode (which is discussed in the following section). Therefore, if you have Flag 8 set (so the 'C' is showing in the display) you should clear it (unless, of course, you're working with complex numbers).

Flag 9 causes the display to blink, which is fun--and can be useful--in programs. Remember, however, that this flag is cleared whenever you turn off your calculator or press [<-].

Try this: As an exercise in using flags, write a program that checks each flag (0 through 9) and generates a number in the X-register that indicates which flags are set. For example, if flags 1, 2, and 4 are set (and the rest are clear), the program will generate the number 124. If flags 5 and 0 are set, the program will generate the number 50. If no flags are set, the program will generate: -1.

Solution: We have two program listings. They both use the same logic, but the second listing uses indirect addressing of flags--to shorten the routine.



Addressing the flags DIRECTLY:

```
001   LBL A
002   FIX 0
003   0
004   F? 1
005   1
006   F? 2
007   2
008   F? 2
009   GSB 9
010   F? 3
011   3
012   F? 3
013   GSB 9
014   F? 4
015   4
016   F? 4
017   GSB 9
018   F? 5
019   5
020   F? 5
021   GSB 9
022   F? 6
023   6
024   F? 6
025   GSB 9
026   F? 7
027   7
028   F? 7
029   GSB 9
030   F? 8
031   8
032   F? 8
033   GSB 9
034   F? 9
035   9
036   F? 9
037   GSB 9
038   F? 0
039   GTO 0
040   1
041   CHS
042   X<>Y
043   X=0
044   X<>Y
045   R/S
046   LBL 0
047   1
048   0
049   x
050   R/S
051   LBL 9
052   X<>Y
053   1
054   0
055   x
056   +
057   RTN
```

Or, addressing the flags INDIRECTLY:

```
001   LBL B
002   1
003   .
004   0
005   0
006   9
007   STO I
008   FIX 0
009   0
010   LBL 1
011   F? I
012   RCL I
013   INT
014   F? I
015   GSB 9
016   ISG I
017   GTO 1
018   F? 0
019   GTO 0
020   1
021   CHS
022   X<>Y
023   X=0
024   X<>Y
025   R/S
026   LBL 0
027   1
028   0
029   x
030   R/S
031   LBL 9
032   X<>Y
033   1
034   0
035   x
036   +
037   RTN
```

The indirect addressing occurs in the loop starting at line 010 and ending at line 017. This loop takes the place of lines 004 through 037 in the first listing. Do you see how useful this indirect addressing can be?

COMPLEX MODE

When you set flag 8, a little 'C' comes on in the display. This 'C' indicates that your calculator is in Complex mode. There are two functions that AUTOMATICALLY set flag 8, thereby putting the machine into Complex mode. These two functions are f I and f Re<>Im.

We can't take the pages needed to describe complex mathematics; the full details of Complex mode on the HP-15C are described very well in the HP manual. But we would like to show you a couple things you need to know about Complex mode.

First, the only time flag 8 should be set is when you are working with complex numbers. Some functions, such as →R and →P, work completely differently in Complex mode. So, be sure the little 'C' is on ONLY when you want it on.

When the calculator is set to Complex mode, it uses some of that "memory used to store other things" to set up an imaginary stack (see pages 71-74 if you don't remember what this memory is):

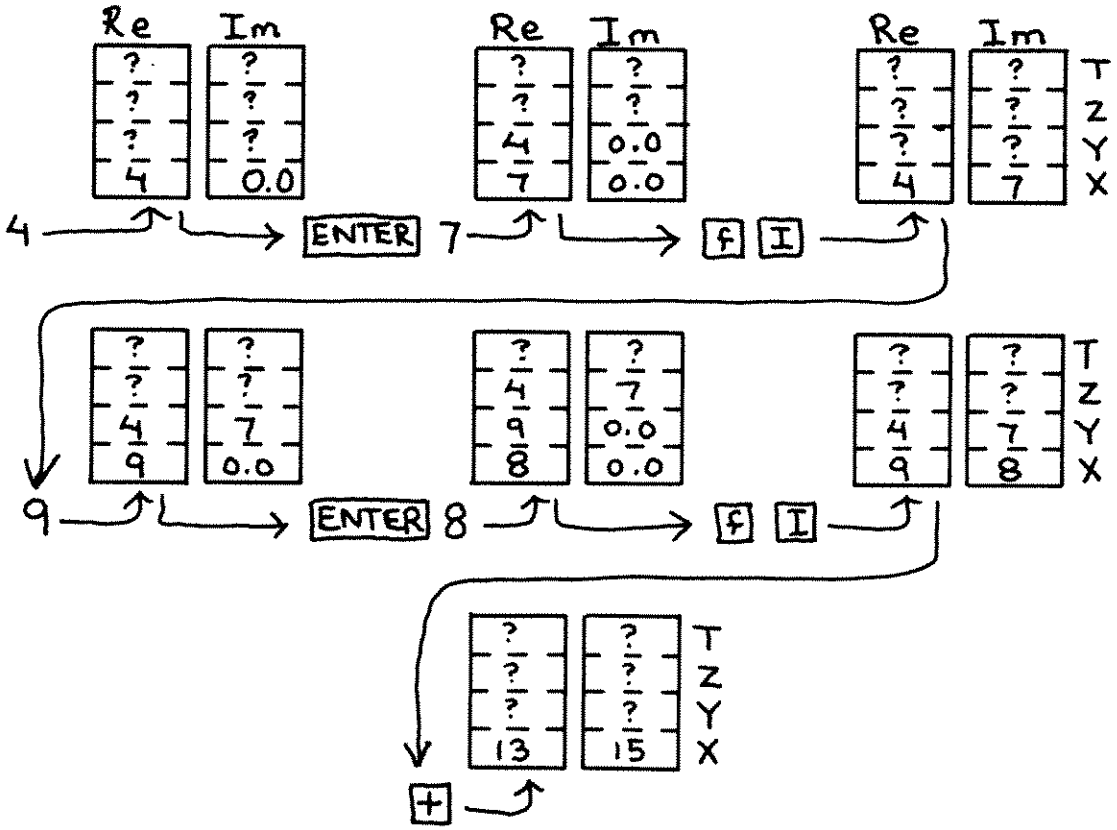
Try this:

$$\begin{array}{r} 4 + i7 \\ +\ 9 + i8 \\ \hline \end{array}$$

Solution: [g] [SF] 8
4 [ENTER] 7 [f] [I]
9 [ENTER] 8 [f] [I]
[+]

Answer: 13 + i15

Here's what happens in the real and imaginary stacks when you go through these keystrokes.



To see the contents of the imaginary X–register, you can either exchange the contents of the real and imaginary X–registers (by pressing f Re<>Im), or you can temporarily bring a copy of the contents of the imaginary X–register into the display (by pressing f (i)).

The important function to understand is f I. Whenever you press f I, the contents of the real X– and Y–registers are combined to form a complex number. These are the details of pressing f I:

1. The contents of the imaginary Z– and T–registers move down one notch.

2. The contents of the real X–register moves into the imaginary Y–register (writing over what was previously stored there).

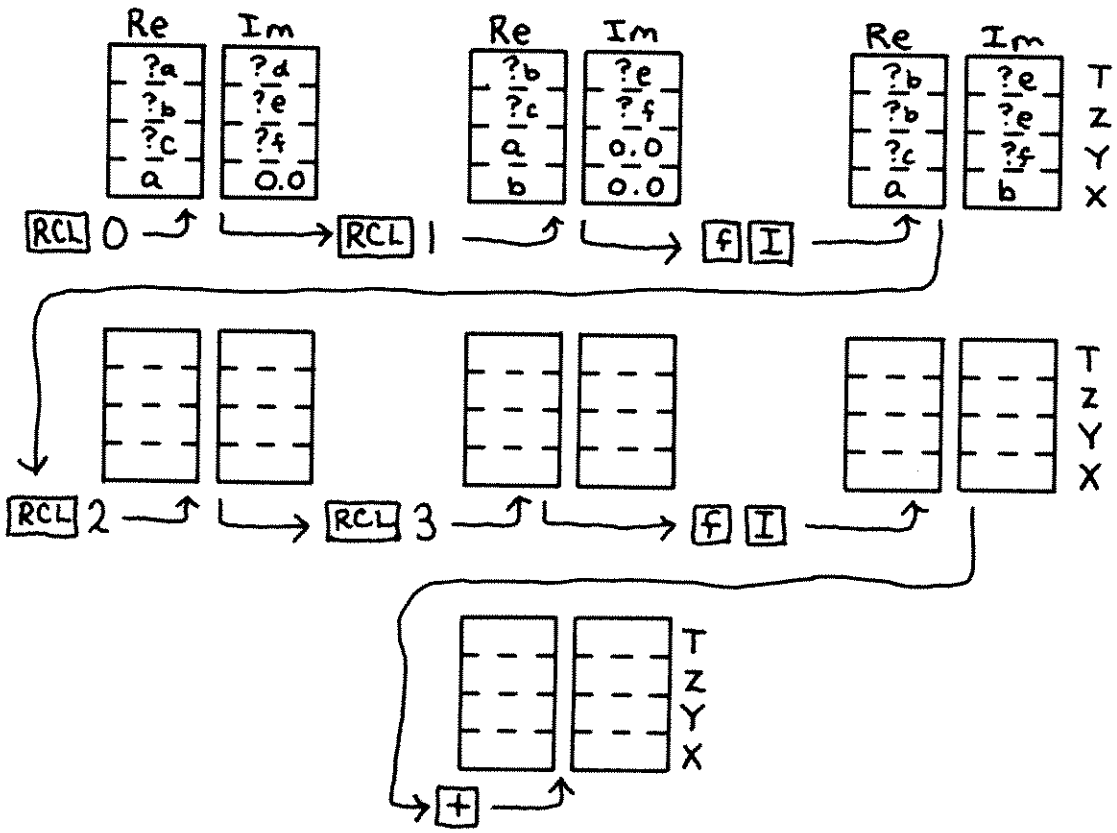
3. The contents of the real Y–, Z–, and T–registers move down one notch.

Try this: Registers 0 through 3 contain the numbers a, b, c, and d respectively. What are the keystrokes to solve the problem:

$$\begin{array}{r} a + ib \\ +\ c + id \\ \hline \end{array}$$

Solution: RCL 0 RCL 1 f I
RCL 2 RCL 3 f I
+

Can you fill in these stack diagrams?



[->R] and [->P] in Complex mode

Just a word to the wise here:

Be aware that the [->R] and [->P] functions use the imaginary X-register in Complex mode as they would use the Y-register in real (non-Complex mode). This is handy for converting complex numbers from "polar" or "phasor" notation to "a + ib" (rectangular) notation, and vice versa.

The "out of Complex mode" operations of these functions are covered in Appendix 2 (page 224).

[SOLVE] and [∫]

The [SOLVE] and [∫] (integrate) functions are two useful functions that we haven't covered in this book--and for good reason:

The manual's descriptions of these functions are excellent, provided you have a strong fundamental knowlege of your calculator. In other words, if you know how the stack works, and how to write a program to evaluate an equation (as you did in the chapter "The Naked Program"), then you'll have no problems in learning to use [SOLVE] and [∫].

MATRICES ON THE HP–15C

This book is not going to cover matrices to any significant extent. But again, the vocabulary and foundation you have gained from this course will aid you in working through the "Calculating With Matrices" section in your manual. The manual's approach is excellent, but go slowly! The computing power of this machine is awesome, and we mortals have to take a little time to grasp it.

To get you started, here is a short summary of the calculator's storage of matrices:

Your calculator stores matrices in the "memory used to store other things" that we discussed on page 73. Each element of a matrix is stored in a separate register. So one three–by–three matrix will use 9 registers of memory; one five–by–five matrix will use 25 registers.

To store these elements, you have to tell the calculator to reserve space for them. This is called "dimensioning" a matrix. But remember this: The HP–15C won't let you dimension a matrix if that would require destruction of a program already stored in the "memory for other things."

Likewise, if you have a matrix stored in that memory, and you then try to key in a program which would require some of that used memory, the calculator wouldn't let you do it (you would see an ERROR message).

With these ideas in mind, you can start studying your manual; observe how programs, matrices, the complex stack, and the SOLVE and integrate functions compete for the memory in your calculator.

Also, notice the different powerful matrix operations (f MATRIX 0 through f MATRIX 9). Notice that these operations are printed in the table on the back of your calculator as reminders.

Appendix 4
Fun Facts to Know and Tell

L.R. and $\hat{y}$,r

Just the names of the functions L.R. (Linear Regression) and $\hat{y}$,r (linear estimate and correlation coefficient) may not give you much of a clue about how to use them.

In fact, like →P and →R, the L.R. and $\hat{y}$,r functions are seldom used to their full extent. So we're going to rattle off a list of things that you can do with these two functions and then show you some examples of how to use them....

1. If you know two points on a line, you can get the equation for a line in the form: $y = Ax + B$, where A is the slope and B is the y–intercept.

2. You can linearly interpolate and extrapolate around two or more points on a graph (or two or more values in a table).

3. You can solve for the slope and y–intercept of a straight line that is the "best fit" through any two or more data points. This is very handy in any kind of experimental environment (such as physics, chemistry, or biology labs) where you are collecting data and then trying to fit that data to a straight line.

So, if you ever find yourself trying to solve something on a piece of graph paper by drawing a straight line through two or more points, chances are you should try one of these functions.

Of course, to use either of them, you have to know a little bit about the statistics registers in your calculator, and you have to know how to key in an X,Y pair for statistical calculations.

Your calculator uses six of the numbered data registers (registers 2 through 7) for these and other statistical functions.

If you want to do anything with statistics on your calculator, the main function you need to know about is the Σ+ function. When you press Σ+, this is what happens:

1. The number in register 2 is incremented by 1.

2. The number in the X–register is added to the number in register 3.

3. The number in the Y–register is added to the number in register 5.

4. The square of the number in the X–register is added to the number in register 4.

5. The square of the number in the Y–register is added to the number in register 5.

6. The numbers in the X– and Y–registers are multiplied together and the result is added to the number in register 7.

7. The number in the X–register is stored in the LSTX–register.

8. The number in register 2 is stored in the X–register.

9. Stack–lift is left disabled.

Whew! That's a lot of operations for just one flick of the finger, right?

Well, to help you remember, items 1 through 6 in the above list are summarized in a table on the back of your calculator. Items 7 through 9 are also useful things to know, but don't WORRY about any of these, because the calculator takes care of them AUTOMATICALLY!

There are really only two things that you need to think about, and they are:

1. Whenever you start a problem that uses the Σ+ function, you need to clear registers 2 through 7. Do this by pressing f CLEAR Σ. (This also clears the stack.)

2. To enter a statistical X,Y data point, you just key in the Y–value, press ENTER to put it into the Y–register, key the X–value in (to the X–register), and press Σ+. The number then appearing in the X–register tells you how many X,Y points you have keyed in.

Try this: A line goes through the points (4, 17) and (7, 32). What is the slope/intercept equation for this line.

Solution: $Y = 5X - 3$

Here are the keystrokes:

f CLEAR Σ	(clears the statistical registers)
17 ENTER 4 Σ+	(accumulates 1st point)
32 ENTER 7 Σ+	(accumulates 2nd point)
f L.R.	

The Y–intercept (–3.00) is now in the X–register, and the slope (5.00) is in the Y–register.

Try this: At a certain temperature, and under a pressure of 230 lbs. per square inch, the specific volume of superheated steam is 2.9276 cubic feet per lb. At that same temperature, but at a pressure of 240 lb per square inch, the specific volume is 2.8024 cubic feet per lb.

At that same temperature, what is the specific volume of superheated steam at a pressure of 234 lbs. per square inch?

Solution: Using linear interpolation, the specific volume is 2.8775 lb per square inch.

This is a typical linear interpolation problem. You are given the "X,Y" coordinates of two points on a line. Then you are given the "X" coordinate of one other point on that line and asked to solve for its "Y" coordinate.

The math is fairly simple to grind out, but it's easier to use the ŷ,r function on your calculator.

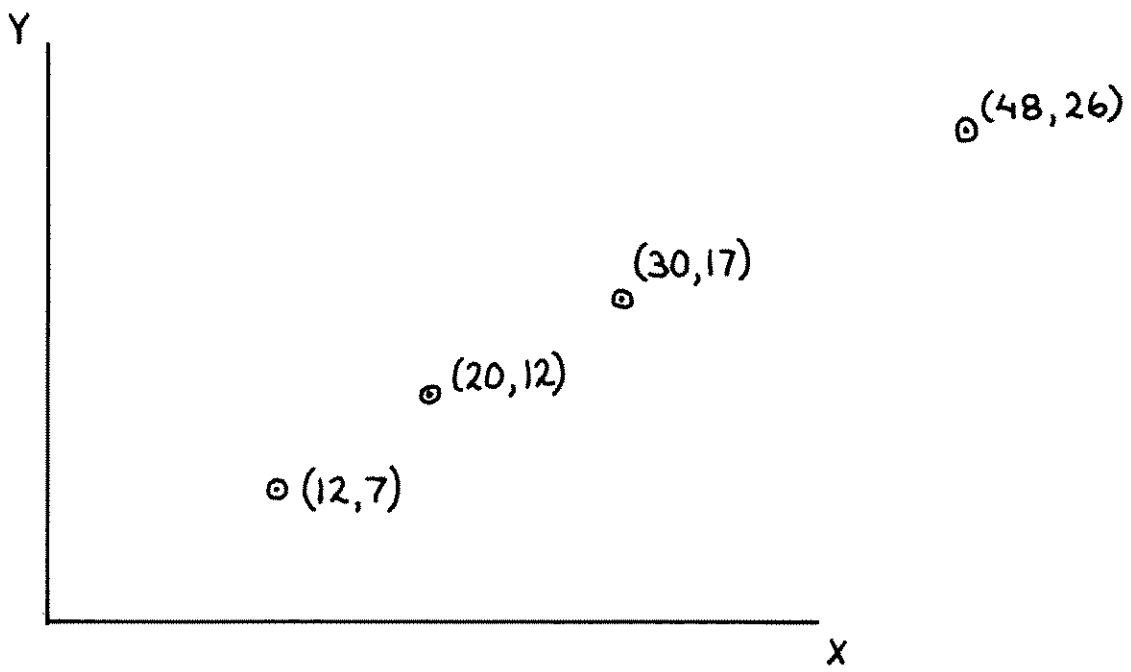
The two points you are given in the above problem are (230, 2.9276) and (240, 2.8024). You are asked to find the "Y" value at X = 234, right?

Here are the keystrokes:

f FIX 4	(so you can check your inputs)
f CLEAR Σ	(to clear the Σ −registers)
2.9276 ENTER 230 Σ+	(accumulate 1st point)
2.8024 ENTER 240 Σ+	(accumulate 2nd point)
234 ŷ,r	(ask for "Y" at a given "X")

That's all there is to it! The answer is in the X−register.

Try this: After a rigorous chemistry experiment, you are faced with a distribution of data that looks like this:



What is the slope of the line that represents the "best linear fit" of this data?

Solution: f CLEAR Σ
26 ENTER 48 Σ+
17 ENTER 30 Σ+
12 ENTER 20 Σ+
7 ENTER 12 Σ+
f L.R. X<>Y

Answer: 0.5214

Mean and Standard Deviation

Two other functions that use the statistical registers are $\bar{x}$ (mean) and s (standard deviation). These functions are easy to use, once you know how to accumulate values in the statistical registers (using $\Sigma+$).

Notice, in the table on the back of your calculator, that both these functions return results for accumulated X–values (into the X–register) AND accumulated Y–values (into the Y–register). Again, remember to press f CLEAR Σ before starting to accumulate statistical values.

→H.MS and →H

We mentioned these functions, briefly, on page 182. They are used to convert from Hours and decimal fractions of hours to Hours, Minutes, and Seconds (i.e., 4.23 hours = 4 hours, 13 minutes, and 48 seconds) and vice versa.

But we didn't mention this:

Hours and Degrees (for measuring angles) use the same base as Hours Minutes and Seconds! You can use →H.MS and →H to convert from degrees, minutes, and seconds to degrees and decimal fractions of degrees!

We hope you enjoyed this book; we certainly enjoyed writing it. In a very real sense, we were just trying to have a conversation with you, because we have found that this simple, one–to–one conversational approach is really very helpful when a person is trying to learn something easily (and retain it well, too).

Unfortunately, some people feel that programming and other technical subjects cannot be taught without using technical jargon. It seems that, after having become comfortable with some "tech–lingo" themselves, these people forget that only computers can truly communicate in computer language.

Well, we TRIED to use plain English here, but if it wasn't as plain in certain spots as it might have been, or if we missed some plottographical errors (or numerical boo–boos), please let us know, so we can keep improving this book. Of course, we would enjoy hearing any other comments you may have about this book, too. Whether your remarks are complimentary or otherwise, we always appreciate it when readers take a moment to let us know how we did. Also, we encourage your suggestions for future books. A handwritten postcard will certainly be sufficient–– and we always read all our mail!

Thanks for the conversation. Good luck!

ALSO AVAILABLE FROM THE PRESS AT GRAPEVINE PUBLICATIONS

--"An Easy Course in Programming the HP-41"

So good that HP uses it to train its own personnel! For beginners or more experienced users; friendly, easy-to-follow lessons on one of the best programmable calculators ever made. Applies to all three models: 41C/CV/CX.

--"An Easy Course in Using the HP-12C and Other HP Financial Calculators"

Also being used by Hewlett-Packard as a training manual for HP employees, this book leads readers to a complete understanding of concepts and methods for financial computations on the HP-12C. Learn all the How's and Why's behind interest, loans, mortgages, investments, uneven cash-flow situations, etc.

--Custom manuscript editing/publishing

--Custom consulting/programming for small computers

--Custom tutorials and 1-day seminars on HP calculators and computers

---

Grapevine Publications, Inc.
P.O. Box 118
Corvallis, Oregon 97339-0118 USA
Tel. (503) 754-0583

TABLE OF CONTENTS

FROM THE PRESS AT GRAPEVINE PUBLICATIONS, INC.

This book will help you understand and feel comfortable using your HP-11C or HP-15C calculator. Its unique, conversational style makes learning to program ENJOYABLE, not intimidating. The authors, both former Hewlett-Packard support enigneers, realize that a relaxed, jargon-free format is the best way to present a technical subject. And if a touch of humor and some delightful illustrations are added, then learning to program your HP calculator becomes both easy and fun!

"AN EASY COURSE IN PROGRAMMING THE HP-11C AND HP-15C" is the easiest and fastest way to master your calculator. Filled with examples, review questions, explanations and fun quizzes, this self-paced book lets you work along at your own rate, learning all the how's and why's of programming. Discover this amazing learning approach, and you'll soon find yourself ENJOYING your calculator!

ISBN 0-931011-02-7