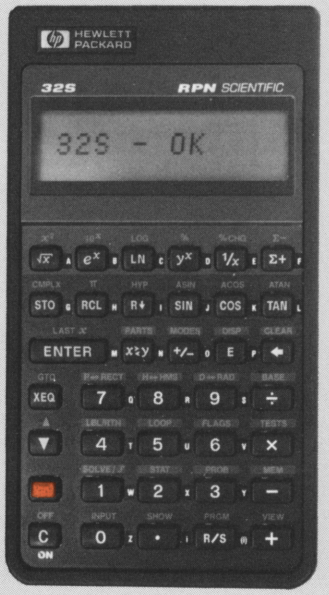
TIPS AND PROGRAMS FOR THE HP-32S

A Handbook for Science and Engineering



by W.A.C. Mier-Jedrzejowicz

A SYNTHETIX Publication

TIPS AND PROGRAMS

FOR THE HP-32S

A Handbook for Science and Engineering

W.A.C. Mier-Jędrzejowicz, Ph.D.

A collection of tips and tricks for users of the HP-32S RPN Scientific calculator, with a set of example programs for science and engineering students.

Includes a table of commonly used constants.

"Tips and Programs for the HP-32S: A Handbook for Science and Engineering"
by W.A.C. Mier-Jędrzejowicz, Ph.D.

First printing September 1988

ISBN 0-937637-05-X

Library of Congress Catalog number:88-62519

Published by:
SYNTHETIX,
P.O. Box 1080,
Berkeley, CA 94701-1080,
USA

INTRODUCTION

Do you own an HP-32S? If so, congratulations!

Or are you looking at this book and thinking of buying an HP-32S? We hope it will help you make up your mind!

The HP-32S is a remarkable calculator - for little more than the cost of an ordinary scientific calculator it gives you the extra power of features such as a function Solver and RPN calculation, together with the quality and prestige of a Hewlett Packard product. As you read through this book you will see how you can use the HP-32S to solve problems, and how you can find additional tricks and shortcuts to make it work even faster and better during your studies and after them.

This book has four chapters, each subdivided into specific points. The first chapter describes tips and suggestions which can help you use the HP-32S efficiently and quickly. Chapter 1 also includes some notes for readers who have already used other HP calculators. These readers will appreciate the comparison of the HP-32S with other HP calculators, and suggestions for adapting programs written for those calculators to the HP-32S. The second chapter shows short programs of the type you might use in a college course - tricks described in the first chapter will be used in these short programs and new tricks will be introduced and explained. The third chapter shows a few longer programs which add extra abilities to the HP-32S statistics and integration functions. These programs are very useful - even if the technical situations addressed do not suit your needs they will show how to write programs for your own studies. The fourth chapter lists further sources of information on your calculator and programming, and other useful products.

A list of commonly used constants in both English and metric units is given on the inside rear cover for those students who want to use this book with their HP-32S in their courses. So far as possible this book avoids using examples that are covered in the book of

Engineering examples produced by HP for the HP-32S.

I want to express my thanks to Tony Collinson and Mark Ellis of HP UK for keeping me up to date on HP matters, and to members of HP user clubs all over the world for their friendship and support. My thanks go as ever to my Mother for helping while I wrote.

Special Keys and Symbols

Some HP-32S keys and symbols are difficult to represent using ordinary printer characters - this page explains how they are shown in the book.

The orange key is used to select the instructions or menus written in orange above other keys. This is like a SHIFT key on a typewriter; it will be called SHIFT. For example, to select the statistics menu you press the orange key first, then the 2 key with STAT in orange above it. Doing this will be written as just STAT.

The left-arrow or back-arrow or delete key deletes the character to the immediate left of the cursor while numbers or long instructions are being typed in, at other times it deletes a program step, clears a message or menu, and if none of these is displayed then it clears the value in the stack X register. I shall usually call this the DEL key or <-.

The symbol for pi (SHIFT .) will just be written **pi**, the symbol for "not equals" will be ≠. The sigma symbol will usually be printed as Σ and the integral symbol will be INTEGRAL. SQRT will be used for the square root symbol. The multiplication symbol used by the HP-32S looks very much like a small "x", so to avoid confusion I shall use "x" for the small letter x, and an asterisk "*" as the multiplication symbol. The SOLVE/INTEGRATE key will simply be called SOLVE.

When a set of HP-32S keystrokes, or a program, is printed then keys which are pressed separately are printed with spaces between them. For example LN 1/x means "press the key marked LN, then press the key marked 1/x". Numeric digits are printed without spaces between them, so 123 means "press 1, then 2 and then 3." If numbers have spaces between them then they are separate numbers, for example 45 6 represents the two numbers 45 and 6. Numbers followed by powers of ten will be printed in the same way as the HP-32S shows them - the number followed by an E and then the power of ten.

CONTENTS

CHAPTER 1 - TIPS

1.1 BEFORE WE BEGIN As you go through this book (or the HP-32S manual) you will sometimes find especially interesting points. You might want to underline them or highlight them so you can easily find them later. A better idea is to keep a pocket notebook with your HP-32S and to keep notes of good ideas in that. The HP-32S is about the same size as a small notebook, so you should have no trouble carrying one around and it will be available whenever you want to write down a new idea or program, or when you want to find an old one.

This book is for anyone who wants some tips for using an HP-32S, but it is particularly for science and engineering students. The HP-32S can show numbers in "Scientific" notation, or in "Engineering" notation, so to be even-handed about it I shall use neither! Answers to examples will be shown as the calculator displays them if it is set to FIX 4. It would be a good idea for you to press the keys SHIFT then DISP, then the FX key (the key directly below FX which is now in the display), and finally 4. That way numbers in this book will agree with the results you see on your HP-32S when you do the examples.

1.2 THINKING ABOUT RPN If you have read the HP-32S manual (you certainly should do this!) you have already met the Reverse Polish Notation (RPN) method of doing calculations. If you have not used an RPN calculator previously then you might find it difficult to understand RPN at first. With experience and after you have done some examples you will get used to it. Here are a couple of helpful ways to think about it.

When you first learned to do addition you were probably taught to do it like this:

To add 123 and 456:

Write down the first number	123
Now write the second number below it	123 456
Now add them up	123 456 +
and write the answer under them	123 456 + ---- 579

This is called "stack arithmetic". You write a stack of numbers with the first one on top, the second one below it, and the answer at the bottom. To do more complicated sums you just use a bigger stack. To do another sum using the answer from the first one you just write the next part of the sum under the previous answer and carry on using the stack. Say you want to take away 321 from the answer above.

Write 321 under the answer you already have	579 321
Subtract it	579 321 -
and write the new answer at the bottom	579 321 - ---- 258

Now try the same calculation on your HP-32S.

Type in the first number	123
press ENTER to get it out of the way (you still see 123, but pressing ENTER has told the HP-32S you have entered the whole number and you are ready to put in a new number)	123
Now type the second number below it (the first number vanishes when you put in the second one)	456
Now add them up (just press +)	+
and you see the answer	579

The HP-32S does the arithmetic in the same way as you would have done it!

Now carry on with the second part of the question:

Type 321 under the answer you already have	321
Subtract it (just press -)	-
and see the new answer	258

Actually this is explained in the HP-32S manual (you <u>have</u> read it, haven't you?), but it helps to see it described as stack arithmetic - often called "stack notation". You can see that the arithmetic operation (plus or minus in this case) is done <u>after</u> everything else has been written down. This might seem odd - we usually say "add 123 and 456", so it is called "reverse notation". Since it comes from a notation invented by the Polish logician Jan Lukasiewicz it is called "Reverse Polish Notation" or RPN, but you can call it

"stack notation" instead - this might make it easier for you to think about - and I won't mind (well, not much, after all, I am Polish).

All very good, you might say, but later on I learned to write things down in "Algebraic Notation", for example:

123 + 456 - 321 = 258

so why go back to the old way? One reason is that this way of calculating always lets you see intermediate answers and check them - for example you could see the 579 after you did the addition. Another is that it lets you correct mistakes because the number you used last is saved and can be brought back with the LASTx key. A third reason is that stack notation makes programs simpler - even computers which use BASIC translate algebraic expressions into stack notation before working them out.

You still might not be convinced and think "reverse notation" is the wrong way to write things down - don't we write SIN(x) or LOG(x)? Not always - the factorial function is written x! with the symbol after the x. If you find RPN difficult just think of these two things - "stack notation" and factorial functions - soon you will get used to it and will appreciate it.

1.3 MOVING AROUND THE KEYBOARD The HP-32S is there to make life easier for you, so you should look for easy ways to use it - and finding a way to do something with the smallest number of keystrokes is one way to make life easier. There are often several ways to carry out the same calculation on the HP-32S. To double a number you can type it in twice and press + (silly!), you can enter the number, then type 2 and multiply (better), or you can type in the number and press ENTER then + (best). If you want a permanent copy of the number you can STO it, and then do RCL+ to double it, but ENTER + is much the simplest way.

Another example is squaring a number. You can type it in, then

press SHIFT and x^2, but pressing ENTER and multiply requires less moving around the keyboard. X^2 is best used in programs, where it takes less memory ("ENTER multiply" and "x^2" both require two keystrokes, but x^2 is stored in a program as one instruction instead of two for ENTER multiply).

Again, to see what is in a register it is quicker to press RCL and the register name, then R↓ (Roll-down, the key with R and a downward arrow), than it is to press SHIFT VIEW register name, and then backarrow to return to the normal display.

In the above cases, different methods to do the same thing can have different effects on the stack. If you have important numbers in the stack you might have to use an inefficient way of calculating something rather than lose a value from the top register of the stack. This is not always the case, for example, subtracting the value in the second stack register (Y) from the bottom stack register (X) can be done by pressing x<>y and then minus, but pressing minus and then the sign change key (+/-) gives the same result, has the same effect on the stack, and can be easier if your finger is at the bottom of the keyboard after entering the number in X.

There is still a difference though, because the number saved as LASTx is not the same in the two cases - in each case the number that was subtracted is saved as LASTx. An important point to note is that the operation +/- does not save anything in LASTx. +/- actually does two different jobs - it changes the sign of a number (or its exponent) while you are typing the number in, or it changes the sign of a number (but not its exponent) if the number is already at the bottom of the stack. The second operation is an arithmetic operation like addition or subtraction, it is stored in programs as a separate step, not as part of a number, and you might expect it to change what is in LASTx, but it does not!

There are plenty more cases where a bit of thought before a calculation can save you unnecessary moving around the keyboard,

wearing out the keys - and your fingers! Here is a last example; to calculate X + SIN(X) you can press SIN then SHIFT LASTx, then plus, or you can press ENTER, SIN and plus. The second saves you one keystroke, which is useful if you are going to do this sort of calculation a lot.

If you have a notebook as suggested earlier, you can write down whichever of the above tips are of interest to you - leave some space so you can add similar ideas of your own. Later on we shall see more advanced ways to save on keystrokes - the idea is always to make things easy for yourself with the HP-32S.

As you become familiar with the HP-32S keyboard you will learn that some keys do many different things. By far the most important is the ON key. Clearly ON turns the calculator on, and combined with SHIFT turns it off - but the "C" on it reminds you that at other times it acts as the Clear or Cancel key - it interrupts lengthy activities or programs and makes the calculator pay attention to the keyboard again. It also cancels special menus and the PRGM setting. Pressing ON and + or ON and - changes the display contrast, pressing ON and y^x starts the self-test, and pressing ON and e^x resets the calculator. Sometimes this will not reset the calculator - for example if the batteries become very low and the keyboard refuses to respond. In such a case you will have to take out the batteries and short-circuit the HP-32S battery compartment terminals with a metal coin, as described in the manual.

1.4 VARIABLES AND THE STACK The RPN stack has four "registers" each of which holds one number. The X register is the one whose contents are normally shown in the display. The next register up is the Y register - it is used in calculations such as + or / and it is used to hold the imaginary part of complex numbers. The next two registers are the Z and T registers. You can think of T as the "Time" register if you think of the whole stack as X-Y-Z-T as in Relativistic 4-vectors in Physics. Another way to think of T is as the "Top" register of the stack - this is an important idea because the top register is always copied when the bottom value in the stack

changes and the values in the stack move down. There is another register used with the stack - the L or LASTx register which contains the number that was in X before X was last changed by a mathematical operation. It is important to remember that CLX and +/- are not treated as mathematical operations and do not change LASTx. STO or RCL operations do not change LASTx, but RCL arithmetic which changes X does copy X to register L first.

These five "stack registers" are always available - their space can never be used for other purposes. All other registers can be used to store "variables", or programs, or statistics results, or temporary values needed by SOLVE and INTEGRATE. When you store numbers in these extra registers the HP-32S calls them "variables" - for example variable C can be used to store the speed of light, or N can be used to store the Avogadro number. It is important to remember that these "variable registers" or "storage registers" are not the same as the "stack registers". In particular, the stack registers X,Y,Z,T,L are not the same as the storage registers called by the same names. The names of the storage registers are really a shorthand for their numbers - storage register A is really register 1, storage register B is register 2, and so on up to storage register Z which is really register 26. The special storage register "i" is really just register number 27, but it is used for "indirect" operations too. If you think you might confuse the names of the stack registers with the storage registers it might be wise to avoid using storage registers X,Y,Z,T,L and it may be best to avoid register I to avoid confusing it with "i".

If you clear a storage register, or store a zero in it, or carry out an operation that puts a zero in it (for example - using 1E-499 STO*n to put a very small number in register n) then the space used by that register is made free. The HP-32S has 27 "flags" which tell it if a number is stored in each of these registers (or variables). If the flag says there is nothing in the register then that register uses no space, and the variable is treated as if it contained a zero (for example if you try to RCL it). When you try to use any of these registers, the HP-32S has to check all the register flags from

1 up to the register you want, to check if each of these registers exists. For every register which does exist, the HP-32S has to allow for the space occupied when looking for registers beyond it. Thus RCL Z is noticeably slower in a program than RCL A, because the HP-32S has to check every register from 1 to 26 in order to find out where register Z is. We'll come back to this in the chapter on short programs.

If you are going to use a variable a lot then it is sensible to use one whose name is near the STO or RCL keys. The best variable name of all is H, since you can recall this just by pressing the RCL key twice! If you are going to use a variable to do a lot of arithmetic then use the same trick - to add a lot of numbers to a variable it might be wise to use variable Y, since pressing STO+Y is particularly easy because Y is near the + key. In fact the (i) key is even closer to + so it might be best to store a register number in i (say 1) and then use STO+(i) repeatedly to add numbers to register A (because this is register number 1). Whenever you do storage arithmetic of this sort, remember to "initialize" the register first - STO a zero in a register before using STO+ to add numbers in it, or STO a 1 in a register if you are going to use STO* or STO/ to multiply or divide it. In any case it is usually easier to use the stack for repeated additions or multiplications, as we shall see below.

1.5 CONSTANTS AND SHORT PROGRAMS Some sorts of work need the use of a constant. For example if you want to change a whole lot of measurements from inches to centimeters then you will want to multiply each measurement by 2.54. You could store 2.54 in variable V, then type in every measurement and press RCL* V, but this takes three keystrokes and a lot of moving around the keyboard. It is much simpler to use the duplication of the top stack register. Type 2.54 and press ENTER three times. You now have 2.54 in the whole stack. Type the first number you want to convert and press * to see the converted result. The number in stack register Y will have been used to multiply the number in X, the number in stack register Z will have moved to Y, the number in T will have dropped down to Z,

but it will also have been copied in T. To convert the next number press the backarrow key (or the ON key) to clear stack register X, then type in the next number and press * again. You can carry on doing this for every number you need to convert - each conversion now takes only two keys (multiply and clear) and both these keys are near the numeric digits on the keyboard.

The same trick works for addition, but if you want to divide by a constant or subtract a constant then you have to use a different trick, involving LASTx. The first time you do a division (say by 2.54 to convert centimeters to inches) you must type in the first number to convert, then press ENTER, then type in the constant (2.54 in this case), then press the divide key. You have your first answer, and the constant is saved in stack register L. To convert the next number just type it in, then press SHIFT, LASTx, divide. You can carry on like this for every number, because dividing by the constant always puts it back in L - the only problem is if you have to do some other calculation in the meantime and lose your constant from L. The same trick will work if you want to subtract a constant repeatedly. With LASTx the number of keystrokes is the same as it would be otherwise, but there is less moving around the keyboard.

Pressing SHIFT, LASTx, divide is three keystrokes - it is not much easier than using RCL-divide-n. There should be an easier trick, and there is! You can write a short program to do the whole calculation. In case you have not written a program before, here are the steps to follow (but you <u>should</u> have written a program by now if you have read the manual!). First store your constant in a variable - let's say you use variable K. Now press SHIFT GTO . . (that's SHIFT, then the XEQ key, then the point key twice. Next press SHIFT PRGM and you are ready to write your program. Begin with a label, so press SHIFT LBL/RTN and then the key below the display marked by LBL and an arrow in the display, then press one of the keys marked A..Z - let's use LBL D (for Divide), so press the key marked with a D to its lower right (that's the y^x key). You will see the first line of your program "D01 LBL D". The next step of the program is to recall and divide by the

constant, so press RCL, then divide and K. The last step is to finish the program by returning to its beginning, so press SHIFT LBL/RTN and then the key marked RTN below the display. To finish writing the program press the ON key, and to get to the beginning of the program press the down-arrow key (the one just above SHIFT). The program you have written should look like this:

```
D01 LBL D
D02 RCL÷ K
D03 RTN
```

You now have a program which divides the value in stack register X by the constant in variable K. To divide a set of numbers by this constant, you can at any time type each number into the HP-32S, then press XEQ D. Actually this still takes two keys, but after you have done it once the HP-32S has found the program for you, and has gone back to its beginning because the program ends with RTN; it is important to have that RTN at the end of your program. The second and subsequent times you can therefore divide by the constant simply by pressing R/S (the Run program and Stop program key). We have reduced the business of dividing by a constant to a simple matter of pressing just one key!

After you have used your HP-32S for some time you will have more than one program in it, and you will only be able to use R/S to run a program if you know that you are already at that program, and if it is the first program in program memory. If it is not, then you can use another trick which will be described in the next chapter. As I said above, you should use XEQ to use the program the first time, then you can re-use it just by pressing R/S. Should you happen to change the value in variable K then the program will divide by a different constant - you might want this if you want to use the same program to divide by a different constant, but you might prefer to write the constant right inside the program, so it will not get changed by accident. In that case replace the program line which says RCL ÷ K with two lines - one containing the number and one containing the divide instruction.

This trick of using a short program to divide by a constant can be used equally well for adding, subtracting, multiplying or even raising to a power. If you want to subtract numbers from a constant, to divide a constant by different numbers, or to raise a constant to different powers, then you must RCL the constant first, do x<>y to exchange the constant with the number you are using, and then do the subtract, divide, or y^x. In all cases though, you will have reduced the whole operation to a single press of the R/S key once your program is written!

1.6 MORE SHORTCUTS The trick of using a short program to simplify an operation can be used in many other cases. For instance if you want to find the hyperbolic arc tangent of several numbers you have to press SHIFT HYP SHIFT ATAN for every number. It would be much quicker to write a program which contains the three steps:

```
H01 LBL H
H02 ATANH
H03 RTN
```

Every time you need to get a hyperbolic arc tangent now you need only press XEQ H, or if you need to calculate several of them then you can just press R/S repeatedly.

The same trick is worth using for any operation which uses more than a few keystrokes. In the next chapter we shall come to some "real programs" which do a complete calculation, but it is entirely reasonable to use programs as described here, just to do one step. Remember that using R/S this way only works for the first program in memory, see Chapter 2 for a way to do the same thing with any program.

A particularly interesting case is the calculation of a vector amplitude:

$$V = SQRT\ (A^2 + B^2 + C^2 + ...)$$

You can calculate this by squaring each number, adding up all the squares, and then taking the square root. The HP-32S does have a vector amplitude function built into it though, so why not use that? Type in the numbers A and B, then use the keystrokes SHIFT P<->R and then press the key marked y,x->Θ,r . This calculates the vector magnitude and direction, putting the amplitude in stack register X. Stack register X now contains

$$X = SQRT\ (A^2 + B^2)$$

To add another vector, just type in its magnitude and repeat the process. If you are calculating a sum of squares for many values you can repeat this for all the values. The problem is that y,x->Θ ,r takes four keystrokes each time, so entering each number, squaring, adding, and taking the square root at the end is just as quick. To get round this you can write another short program:

```
V01 LBL V
V02 y,x->Θ,r
V03 RTN
```

Now the business of adding up a vector sum of squares is reduced to pressing XEQ V repeatedly (or just R/S after the first time). If you really want the actual sum of the squares then you can finish by pressing ENTER and multiply to square the final result! You could get a sum of squares by using the summation operations instead and then using the STAT menu to recall the sum of x-squared. Unfortunately this uses six storage registers all at once, and you have to remember to clear the statistics registers before you begin.

The operation y,x->Θ,r can do another useful thing - it can calculate the complete arc tan of x and y. For example ATAN(y/x) where x is -.3 and y is -.6 is -153.435 degrees, but if you calculate -.3/-.6 and press ATAN then you get the answer in the "first quadrant" - namely 26.565 degrees. To get the answer in the

correct quadrant you can type in y, press ENTER, then type in x, then use y,x->Θ,r and finally press the roll-down key to see the calculated angle. If you have written the program above then you can just press R/S and R↓ to get the results, and if you plan to use this repeatedly just to get a complete arc tangent then you should put the extra step R↓ after step 2 of the program.

The opposite operation Θ,r->y,x can be very useful too - for example if you need the sine and the cosine of an angle then you can type in the angle, put 1 in the X register, and use Θ,r->y,x to get the sine in Y and the cosine in X. Once again, if you are going to use this a lot then it is worth writing a very short program to avoid having to press lots of keys.

There are more operations which can be used for extra purposes. An example is that both 10^x and the factorial function can be used in programs to replace a zero with a one. In section 2.3 we shall come to another important use of 10^x to save space in programs.

1.7 FLAGS, MODES AND TESTS The HP-32S has seven "flags" which you can set, clear or test. A flag can be treated as a number; 0 or 1, or it can be thought of as a "signal flag" which is raised (set) or lowered (cleared). A flag can also be thought of as the answer to a question; yes or no - or true or false. Flags 5 and 6 have special meanings - flag 5 is the answer to the question "should overflows stop a program?" - if flag 5 is set then the answer is yes. Flag 6 answers the question "has an overflow occurred since the last time flag 6 was cleared?" - if flag 6 is set then the answer is yes.

Here is an example of using flag 6. If you are using factorials to calculate the expression:

$$\frac{x!}{y!}$$

and one of the factorials overflows then you might get some odd

results - say you try to calculate:

$$\frac{254!}{253!}$$

It is obvious that the answer should be 254, but 254! is just bigger than 1E500; it causes an overflow and the HP-32S gives the answer 1.9329, not 254. If you write a program to calculate the formula above you can test flag 6 immediately after the calculation to check if the result is likely to be wrong. You should of course clear flag 6 before the calculation.

This is another case where you could use a different function - in this case you could replace

$$\frac{x!}{y!} \quad \text{with} \quad P_{n,r}$$

where n = x, and r=x-y. The formula for Pn,r (permutation of n objects taken r at a time) is:

$$\frac{n!}{(n-r)!}$$

The point is that Pn,r is calculated in such a way that it will not overflow if its numerator or denominator overflow - it only overflows if the final result is greater than or equal to 1E500. In the example of 254!/253! you would use Pn,r with 254 in stack register Y and 1 in stack register X and you would get the right answer, namely 254.

You can use flags 0 to 4 in the same way, to answer questions. If you are working in degrees Celsius or Fahrenheit you can set flag 0 to show you are using Celsius and clear it to show you are using Fahrenheit. A program could test the flag and use different formulae depending on which is true. A setting like this is called a "mode" - flag 0 can be used to answer the question "am I in Celsius mode or Fahrenheit mode?" The display shows if any of flags

0 to 3 are set, and the little 0 is well suited to displaying the temperature mode, as it looks like the degree symbol.

The HP-32S can also be set to one of three trigonometric modes - DEG, RAD or GRAD. On some calculators a pair of flags is used to let programs find out which mode is set - on the HP-32S you have to use a short program instead. For example the program below sets flag 1 if you are in RAD mode, and clears it otherwise:

Program step	**Comment**
T01 LBL R	Label "R" to check Radians mode
T02 CF 1	Assume Radians mode is not set, so clear flag 1
T03 PI	Put PI in register X
T04 COS	COS(X) - on the HP-32S COS(PI) is exactly -1 in RAD mode
T05 1	put 1 in X
T06 +	and add it
T07 x=0?	result is zero if RAD mode is set
T08 SF 1	if result is zero, then set flag 1
T09 R↓	roll down the result so the original value is put back in X
T10 RTN	finish the program with a RTN

(The checksum of this program is 7010.)

You can use this program to check for RAD mode, and a similar program to check for GRAD mode (if neither is set then you must be in DEG mode!). In fact it is usually easier to simply set the mode you need.

REMEMBER! If you are doing some trigonometric work you must make sure that you have set the correct trigonometric mode first. It is no good at all handing in a completed piece of project work, one minute before the deadline, and then remembering that you have done all your calculus with the HP-32S set to DEG mode!

Checking the display mode (FIX, SCI, ENG or ALL) and the base mode

(BIN, OCT, DEC or HEX) from a program is much more difficult. Checking the number of digits displayed can be done with a program which rounds a number and then compares it with the original, but that is not easy either. Fortunately there are only a few times when you need to check one of these modes from a program.

The above program uses the test x=0?. The HP-32S provides one flag test, four tests to compare X with zero, and four tests to compare X with Y. If you need one of these tests you can rearrange a program to use the test which is available - sometimes you need to use a GTO and a LBL to jump around a piece of program. To create the test X>=0? (i.e. "is X greater than or equal to zero" which is the opposite to X<0?) you can use the two program steps X≠0? X>0? which have exactly the same effect in a program. The step following the second of these two tests is carried out only if X is either zero or greater than zero. You can use the same combination when testing X against Y. This tip and many others like it were originally suggested for older HP calculators such as the HP-67 or the HP-41 and they are described in some of the books listed in Chapter 4.

Some calculators let you do tests "from the keyboard" - for example on the HP-41 you can press the X=Y? key just as if you were doing a calculation and the result "YES" or "NO" is shown in the display. This is much simpler than looking at the number in X, using x<>y to see Y, then if they look the same using SHOW to see all the digits of both, and remembering to use x<>y a second time to put X back where it was. The HP-32S tests do not give any result if you try to do them like a keyboard calculation - they are only useful in a program, where they follow the "do if true" rule - the next program step after the test is done only if the test was true. If you do want to do a test "from the keyboard" then here are some tips.

First of all, you can see if flags 0, 1, 2 and 3 are set because their status is shown in the display - you do not need to do any test to check if they are set or clear! Flags 4, 5 and 6 are not shown in the display, so you should avoid using them if you want to see their settings. In fact, if you are writing a game program then

you can use flags 0, 1, 2 and 3 to show what is happenning, and you can use flag 4 to keep a note of something that is to be kept a secret.

Secondly, you can always see if X is negative, zero, or positive, since it is shown in the display. That leaves the tests which compare X with Y. One trick is to press the minus key! If the result is zero then X and Y were equal, if the result is negative then X was larger, and if the result is positive then X was smaller. You can then use LASTx and add to get the original value of Y, and LASTx again to get the original value of X. If you prefer, you can write a program which sets flag 0 if X and Y are equal, sets flag 1 if X is greater, and clears both flags if Y is greater.

The flags 0, 1, 2 and 3 can be used for other purposes too. Here is a last example - if you write a long program which should not be interrupted while it is rearranging some variables then you could set flag 1 (the 1 looks like an exclamation mark) while the rearranging is going on, to warn yourself not to interrupt the program (with ON or R/S) at that time. When the important part is finished, the program can clear the flag again.

1.8 PSE/VIEW/STOP/RCL/INPUT Each of these five instructions can be used to show a number, but you have to make sure you choose the right one. PSE is the simplest - it stops a running program for 1 second and lets you see the value in the stack X register. The little PRGM annunciator in the display stays turned on to show that a program is still running. After 1 second the program carries on running. If you want to see a number for more than a second then you can put two or more PSE instructions one after another. PSE is useful in programs which slowly work their way through many iterations - it can show the iteration number, or the latest answer, and you can decide to stop the program when you are happy with what you see.

PSE is only useful in a program, because while you are doing

keyboard calculations you can see what is in X anyway. If you want to see a variable while doing keyboard calculations you can press VIEW and the name of the variable. The variable is shown in the display, and you can show all of its digits by pressing SHOW, but it does not replace X, and you can show X again by pressing the backarrow or ON keys. In a program, VIEW shows the variable, but it stops a program instead of just pausing. This lets you use SHOW to see the whole mantissa. You can also press ENTER to copy the number into X, or CLEAR to clear the variable (both of these work from the keyboard too, not just in programs), or ON, or backarrow to see X. You can then carry on with the program by pressing R/S, or you can go on through the program one step at a time by pressing the down-arrow key.

If you want to see a variable without stopping a program then you have to use RCL and PSE. One problem with this is that you are not told which variable you are looking at - you have to be told by the person who wrote the program, or you have to find out by looking at the program! Remember to use roll-down after the PSE to get the original values of X, Y and Z back to their places (the original T will be replaced with the value you recalled). If you just want to see the number in X then of course you can use STOP on its own.

INPUT is the most complicated of these instructions - in a program it stops the program and gives you a chance to see the variable, SHOW it, change it, or even do calculations with it. This is very much like VIEW, but INPUT also brings a copy of the number to X (and if you change the variable then the copy in X is changed as well). INPUT can only be used in a program, and the INPUT annunciator is turned on so you can tell you are using INPUT. When you use SOLVE it checks each INPUT instruction and skips the INPUT of the variable for which it is solving.

The HP-32S does not let you display messages - VIEW and INPUT are the best you can do, and all they show is a one-letter variable name. One way to show slightly longer messages is to to use VIEW or INPUT together with a hexadecimal number! For example, to say that

a result is bad you can store the number 2989 in register R, then set HEX base mode, and VIEW R. You will see the helpful message "R= BAD" which will certainly tell you something - presumably that the result is bad!

Alternatively you could use INPUT R and see "R? BAD". This could be a question asking if the result is bad, and you could enter 1 R/S if the result is indeed bad, or 0 R/S if the result is good. The program could check whether the answer was 0 or 1, and if not then it could repeat the question. You can make up many other words using the hexadecimal digits A, B, C, D, E, F and 0 (for O), but how many will make meaningful messages? BAD is quite understandable, but what word would you use to ask if a result is good - the best I can think of is ACE!

Even if you cannot make many useful messages this way, you can certainly use this trick to make up some game-playing programs. The words CEDE and DEAD come to mind at once! (Such as in "U= DEAD" at the unsuccessful end of a game!)

Once we are on games - here is an extra tip. You can hide the meaning of a message by storing the message as decimal number, or as a negative number in hexadecimal, so that the text of the message is unclear until you change its sign by using +/- and display it in HEX mode.

1.9 THE HP-32S AND OTHER CALCULATORS The HP-32S design follows in the line of earlier RPN calculators made by Hewlett Packard. This means that many programs written for earlier HP calculators will work on the HP-32S with little change or even none. Tricks used on other RPN calculators will often work on the HP-32S as well. Chapter 4 gives the titles of some good books about RPN in general and about some older HP calculators - if you want to learn more about RPN and to get the best from your HP-32S it would be a good idea to look at some of those books.

The next few paragraphs compare the HP-32S with some previous HP

calculators; some of the tips and programs already described come from such earlier calculators. I shall also compare the HP-32S with some new HP calculators which use algebraic notation. In general terms, the HP-32S uses some new technology and new internal programming, so it provides results with 12 digits, instead of 10 digits as on most previous HP calculators - it also works 5 to 10 times faster than earlier calculators such as the HP-41 and the HP-15C.

The HP-32S is more similar to the HP-15C than to any other previous HP calculator. The HP-15C is an advanced scientific calculator with most of the same mathematical functions as the HP-32S, with SOLVE and INTEGRATE functions, and it is able to work with complex numbers. Like the HP-32S, the HP-15C uses an "i" register for indirect access to registers. Thus nearly any keyboard calculations you can find for the HP-15C will work for the HP-32S.

Many programs written for the HP-15C will also work on the HP-32S, but there are some important differences in the use of programs and of "advanced features". First of all, programs were stored in the HP-15C as a series of "keycodes", not as the names of the instructions, but programs published for HP-15C users generally give the names of functions used, as well as the keycodes, so you should be able to type an HP-15C program into the HP-32S without needing to translate keycodes. The HP-32S does not have the matrix instructions of the HP-15C, but you can do some matrix calculations using the example programs in the examples section of the HP-32S manual. HP-32S complex arithmetic is done using the normal stack, not with a special "complex" stack as on the HP-15C, and not all the HP-15C complex functions are available on the HP-32S. The HP-32S does not have as much memory as the HP-15C, so very long HP-15C programs might not fit into an HP-32S. The HP-32S does not use negative numbers in the "i" register to jump directly to a program line number, as did the HP-15C (the older HP-67 did something similar, called "rapid reverse branching" - the HP-32S does not do this either). SOLVE and INTEGRATE cannot call one another on the HP-32S, so you cannot SOLVE a function which contains an integral.

Actually, you can SOLVE such a function, by writing your own integration program. On the HP-32S SOLVE and INTEGRATE expect to recall the unknown variable, whereas on previous models they expected to find its value in the stack. The general rule is that the more complicated an operation, the less likely it is that the HP-32S will do it like previous calculators. In the end, you might well have to check the manual for the calculator for which the program was originally written.

The above might seem to suggest that the HP-32S compares poorly with the HP-15C - but the HP-15C is much slower than the HP-32S (and until recently it was much more expensive). It would be fairer to compare the HP-32S to the HP-10C and the HP-11C, which were less powerful versions of the HP-15C - they did not have the advanced features of the HP-15C but otherwise were similar, and originally they cost much more than the HP-32S. The HP-32S has all the functions of the HP-11C, so any HP-11C program should work on an HP-32S. Two other "series 10" calculators of the older style, the HP-12C and the HP-16C were specialist financial and computer science models. The HP-32S can display numbers in binary, octal and hexadecimal bases, but it is not designed to do logical operations such as AND, OR, SHIFT, ROTATE, which the HP-16C does. The HP-32S can be programmed to do a few financial operations like the HP-12C, but there is a different new HP model in the same price range as the HP-32S, designed specifically for financial work.

The other calculator similar to the HP-32S is the HP-41. The HP-32S copies HP-41 functions, as well as features such as displaying a function name if its key is kept down, or displaying function names, instead of keycodes, in programs. Keeping a key pressed down on an HP-41 for more than a short while cancels that function - unfortunately the HP-32S does not do this - HP-41 users (indeed all users) who use an HP-32S must be very careful not to press the wrong key because there is no way to cancel a function key once it has been pressed.

There were actually three HP-41 models, and the HP-41 has many

features that allow it to be extended to act as a pocket computer, able to control external equipment. The HP-32S does not copy any of these features - its similarity to the HP-41 is restricted to the fact that it has similar calculation and programming features. This does mean, however, that HP-32S users can take advantage of the large number of solution books and programs that are available for HP-41 users. If you need a collection of programs in Physics, Chemical Engineering, or Optometry, for example, then you can buy the HP-41 Solution book for that subject, and adapt the programs for your use on the HP-32S. There are a few important differences - the HP-41 can RCL and STO into stack registers as well as storage registers, any register can be exchanged with X, and any register can be used for "indirect" control - there is no special "i" register. The HP-41 has a separate "alpha" mode and can create messages made of words - much more versatile than INPUT. On the other hand the HP-41 does not provide complex arithmetic or hyperbolics (except on plug-in extension modules), nor RCL arithmetic, and it is much more expensive than the HP-32S.

The HP-41 was followed by the HP-71B, a very advanced (but expensive) handheld computer which uses the BASIC language, and can be extended in many ways by the addition of plug-in modules. One feature of the HP-71B of interest to HP-32S users is that a plug-in module for the HP-71B allows it to run HP-41 programs, so some HP-41 style programs, which can be used on the HP-32S, have been written for the HP-71B. The chip which controls the HP-32S was first used in the HP-71B. Since then Hewlett-Packard has announced a whole range of calculators which use the same chip. The HP-18B (Business Consultant), HP-19B (Business Consultant II), and HP-17B are specifically business calculators, and another low-cost business calculator is to be announced soon after the HP-32S. The HP-28C and HP-28S are very powerful technical calculators which use RPN, like the HP-32S, but their RPN command set is more like the computer language FORTH (available for the HP-71B too), so adapting their programs to the HP-32S is not always easy. Nevertheless, if you find a program for one of these calculators which does a job you need, you should be able to modify it for your HP-32S - if you find

this difficult then you might find someone to help you in your local HP users' club (see chapter 4). Many clubs also have a library where you can find a manual for the older calculator so you can see what needs changing. One important feature common to all these new calculators is the SOLVE function, which works in a similar way on all the different models.

There are also three new technical/scientific HP calculators which have the Solver but which use algebraic notation. These are the HP-27S, the HP-22S and the HP-20S. There is some common ground between these models and the HP-32S, but not much (although all have the same design and look the same from a distance!). Programs written for one sort have to be translated to work on the other sort, but there will be times when this is worth the trouble. For example the HP-22S (the algebraic model most similar to the HP-32S in price and features) comes with a range of commonly used equations built into it; you may find it worthwhile to borrow an HP-22S manual and see if it is worth writing RPN versions of some of these equations for your own use.

SOLVE and INTEGRATE were first provided as built-in commands on the HP-34. Many programs and keyboard calculations for the HP-34 and other HP calculators made before the HP-41 and the HP-15C will work on the HP-32S as well. If you are trying to solve a particular problem it is worth looking for a solution designed for use on one of these older RPN calculators.

1.10 ADVANCED FEATURES AND NON-STANDARD COMMANDS The SOLVE and INTEGRATE functions are much more complicated and advanced than functions like + or SIN. When you first use them you can just follow the instructions in the main part of the manual but later on it is worth reading the extra information in the appendices and using some extra tricks.

One useful trick is to modify the program you are solving or integrating so it will provide extra information. For example, to

see how quickly SOLVE is approaching 0, put a PSE at the end of the program which works out the function whose value is to be zero. Every time the function is calculated, the program will stop for a second and show you the value it found. If the value gets close to zero you can stop the program without waiting for SOLVE to find an exact result. The same program could display the value of the unknown variable, by recalling it and doing a PSE. Another trick is worth using if you think an exact solution is difficult to find, but an approximate one is good enough. Decide how close to zero you want to get - say you think any function value smaller than 2E-7 is good enough. Then change your program so it will return zero if the value it has calculated is smaller than this limit. The following lines at the end of a program will do the trick:

Program	**Comment**
x=0?	If the result is exactly zero
RTN	then there is no need to check it, return at once
ENTER	Save a copy of the exact result
ABS	A result on either side of zero is OK
2E-7	The limit below which a solution is acceptable
x>y?	Are we below the limit?
0	If so, put in a zero
x≠0?	If the result is not zero, then recover the original value
R↓	by rolling down the stack
RTN	Return to finish this calculation

Instead of comparing the result to a specified limit, you could just select a suitable display mode and round the result. BE WARNED: this particular tip will not always work. For example the above will claim to find a zero of the function:

$$x^2 + 1E\text{-}7$$

at any value of x between x=-SQRT(1E-7) and x=SQRT(1E-7), whereas the function actually does not have a zero anywhere. The point of this tip is that it lets you speed up the search for a zero.

When using SOLVE in a program you should allow for the fact that SOLVE acts like a do-if-true test function. If a solution was found then the next program step is carried out, if no solution was found then the next step after SOLVE is skipped. This allows you to GTO the next part of your program if SOLVE was successful, but stop or try again if SOLVE failed to find a solution.

Advanced calculators like the HP-32S have the potential to create instructions which are not described in the manual. Such "non-standard" functions can be very useful - or very dangerous! If you want to learn about them or to share your own discoveries then get in contact with a user club (see Chapter 4).

1.11 TAKE CARE! The HP-32S is a robust calculator - it can survive physical maltreatment and it has functions that let you correct program mistakes - but there are limits! The HP-32S will usually survive a drop of as much as one meter, but the physical shock might lead to a MEMORY CLEAR. If you decide to pour a drink over the keyboard then make sure it does not have sugar in it. Sugar will dry out and will then make keys very "sticky", so avoid it. If you are likely to pour things over your HP-32S, or to use it in wet, sandy or dusty conditions then put it in a sealed plastic bag, or two plastic bags one inside the other.

Be aware of conditions which can give unexpected results - overflow or underflow are particularly insidious. Remember that keys can be previewed, but cannot be cancelled once they have been pressed - pressing another key before releasing the first one does not cancel the action of the first one - it just forces the action of the first one to be carried out. If you are likely to confuse the stack registers X,Y,Z,T,L or the indirect register "i" with variables of the same name then avoid using those variables. Finally, be aware that the HP-32S is a calculator - it will calculate what you tell it to, but it is your business to make sure you give it the right numbers and the right formulae. You are the student, and you are supposed to know that the mass of a proton is 1E-27 kg, not 1E27 kg.

CHAPTER 2 - SHORT PROGRAMS

2.1 TYPING AIDS We have already seen that very short programs can be used to help in keyboard calculations. Let us begin this chapter with three more examples of short programs. First take Einstein's famous equation:

$$E=mc^2$$

You might want to use this to find the energy equivalent of several nuclear particles. Instead of typing in each particle's mass, then pressing ENTER, typing in the speed of light, pressing SHIFT and x^2, and then pressing multiply, you could use the program:

Program step	**Explanation**
E01 LBL E	Label E - for "Energy"
E02 RCL*C	Recall variable C (speed of light) and multiply
E03 RCL*C	Same again to multiply by c^2
E04 RTN	End the program - go back to top of program memory

Note: make sure that "c" is stored in register C, and that it has the correct units!

The first time you write this program, you have to get to its beginning. You can do this by pressing SHIFT GTO E, or SHIFT GTO .E01. If you have put the program at the top of program memory then you can press GTO . . instead, so long as you do not put another program at the top of memory. Immediately after you finish typing in a new program which finishes with RTN and is at the top of memory, you can leave program mode (press ON, or SHIFT PRGM), then press the step-down key "v", above the shift key - this takes you to the top of program memory, and therefore to the beginning of the program. Once you are at the top of the program you can run it by pressing R/S, or you can check it out by single-stepping through it - by pressing the step-down key "v" repeatedly, seeing each program

step in the display, and then seeing the result when you let go of the key.

After you have checked the program, you can get back to it with GTO E, or use it at any time with XEQ E. After using the program once you can use it again just by pressing R/S so long as it is the program at the top of program memory. If you want to use R/S to re-use a program which is <u>not</u> at the top of program memory, then you have to finish it in a different way, as below.

Program step	**Explanation**
E01 LBL E	Label E - for "Energy"
E02 RCL*C	Recall variable C (speed of light) and multiply
E03 RCL*C	Same again to multiply by c^2
E04 STOP	Stop the program and display the result
E05 GTO E	Go back to LBL E, to repeat the calculation

Instead of finishing with RTN, this program stops after calculating the result, then goes back and repeats itself if you press R/S.

A program like this is really a "typing aid", as the title of this section says. The program does no more than save you the effort of typing "c", x^2 and * many times over. This may seem to be a very menial task for a program, but it makes life easier, and that is what all programs should do!

Let us look at another typing aid. Say you are using the statistics registers to fit a straight line to a set of measurements. This does not in itself need a program - you first clear out any previous statistics values with the CLΣ key in the CLEAR menu, then you use the + key to add each measurement to the statistics list. Once you have put in all the measurements you use the STAT menu to select the Linear Regression commands, and work out your straight line. This sounds easy until you try it in a real lab! Then you will find you have to take extra measurements, turn equipment on and off, talk to other students or to a supervisor, and so on. The obvious result

will be that you enter a few readings, then get interrupted, and forget how many readings you have entered. If you are lucky then the sequence number of the last data point is still in the display - but it might not be! Never fear - you can use the "n" command to find out how many measurements you have entered so far - then carry on with the next one. The trouble is, finding the command "n" can take quite a while. So.... write a program to do it for you:

Program step	**Explanation**
N01 LBL N	Label N, as this program finds "n"
N02 n	Get "n", number of readings - in the STAT menu, submenu
N03 STOP	Stop and show this number
N04 GTO N	Go back and repeat this if you need it again

You can use this repeatedly by pressing XEQ N, or simply R/S, and you will still be positioned at the program if you calculate a linear regression for some measurements, then add a few more measurements. (Using a command like Linear Regression does not change your position in program memory.)

You can use this idea for your own purposes, whatever your use of the HP-32S. Here is a third example. You might want two programs which set DEG and RAD mode if you swap between them often - if you write two programs then each one will save you one keystroke. Actually, if you swap between DEG and RAD only, then it would be simpler to use a combined program, as shown on the following page, in the right hand column.

Program to switch from DEG to RAD

```
R01 LBL R
R02 RAD
R03 RTN
```

Program to switch from RAD to DEG

```
D01 LBL D
D02 DEG
D03 RTN
```

Program to switch between RAD and DEG

```
S01 LBL S
S02 90
S03 COS
S04 x=0?
S05 RAD
S06 x≠0?
S07 DEG
S08 R↓
S09 RTN
```

The first two programs set RAD mode and DEG mode and take two keystrokes each (XEQ R) instead of three keystrokes (SHIFT MODES RD). Saving just one keystroke is rarely worth the trouble, but there might be times when it is. The third program switches from one mode to the other, without knowing what the present mode is. It can therefore be used as a subprogram by other programs which need to switch modes, as well as from the keyboard. Even if you decide the saving on keystrokes is not worth the trouble the program does show several useful ideas:

1. Save labels by using one label for a program which does two jobs.

2. Use two alternative tests to select one of two alternative actions.

3. Use R to restore X, Y, Z to their original values.

That should be enough about typing aids - the idea should be clear by now.

2.2 A SIMPLE PENDULUM Let us (at last) look at a typical problem. The period of a simple pendulum is given by the formula

$$p = 2*pi*SQRT(l/g)$$

If "l" is the length of the pendulum in meters and "g" is the acceleration due to gravity in meters per second squared then this formula gives the period "p" in seconds. Here is a short program which asks you for the length of a pendulum and gives its period:

```
P01 LBL P      Label for pendulum period program
P02 INPUT L    Ask for pendulum length (answer must be in meters!)
P03 9.8        Approximate value for g in meters per second squared
P04 ÷          Calculate g/l
P05 SQRT       Calculate square root of g/l
P06 PI         Get pi
P07 *          Multiply by pi
P08 2          Get 2
P09 *          Multiply by 2
P10 STO P      Put the answer in variable P
P11 VIEW P     View P - a suitable name for the period
P12 RTN        Finish the program
```

Program length = 26 bytes
Checksum = 6043

Say you want to find the period of the pendulum in your grandfather clock, whose pendulum is 30 cm long. Do XEQ P, and at the question about L, type in 0.3 and press R/S. The period comes out at 1.0993 seconds (to 4 decimal places).

That's fine, and the program works - but what if you now wanted to work out the length that the pendulum should have to give a period of exactly 1 second? The HP-32S gives you several ways to find the answer. One way would be to try different values near 0.3 until you got a period of 1 second. This can take a long time, but in any case the HP-32S Solver is designed to do exactly the same, and much faster. You want to find a value of L such that P is exactly 1. This means you want to find L such that:

P - 1 = 0

You can rewrite the program above to recall L instead of just asking for it, and to calculate P-1

P01 LBL P	Label for pendulum period program
P02 INPUT L	Ask for pendulum length (reply must be in meters!)
P03 RCL L	Get pendulum length for the calculation
P04 9.8	Approximate value for g in meters per second squared
P05 ÷	Calculate g/l
P06 SQRT	Calculate square root of g/l
P07 PI	Get pi
P08 *	Multiply by pi
P09 2	Get 2
P10 *	Multiply by 2
P11 1	Get the number 1 (the period you want)
P12 -	Subtract from period to get number which should be 0
P13 RTN	Finish the program

Program length = 27.5 bytes
Checksum = DCAE

Now you can use this program with the Solver to find the exact value of L that you want. Press SHIFT SOLVE and then FN and P to say you want to solve the program (function) P. Then press SHIFT SOLVE again and SOLVE L to say you want to solve for the independent variable L. If you still have the value 0.3 meters in L then the display will show the message SOLVING for only a few seconds, and then the answer 0.2482 will be displayed. The pendulum should be 0.2482 meters long if the grandfather clock is to run accurately. The Solver ignored the INPUT L line; L was the variable to be solved for, so the Solver provided its own guesses for L. The Solver uses the program P many times while looking for a value of exactly zero, but if it did need to ask for L, it would do so only once, and would then ignore the line INPUT L.

This shows how the Solver can be used, but the program is no longer

suitable for calculating the period! Instead of writing a special program to solve for the length only, we could use a general program which calculates P or L by using the formula:

$$2*pi*SQRT(L/g) - P = 0$$

This formula can be used to select a value of L (for example 0.3 m as above) or a value of P (for example 1 when we wanted a period of 1 second). Then the Solver can be used to find the other value.

P01 LBL P	Label for pendulum period program
P02 INPUT L	Ask for pendulum length (reply must be in meters!)
P03 INPUT P	Ask for period (reply must be in seconds!)
P04 RCL L	Get pendulum length for the calculation
P05 9.8	Approximate value for g in meters per second squared
P06 ÷	Calculate g/l
P07 SQRT	Calculate square root of g/l
P08 PI	Get pi
P09 *	Multiply by pi
P10 2	Get 2
P11 *	Multiply by 2
P12 RCL P	Get the period
P13 -	Subtract from calculated period to get number which should be 0
P14 RTN	Finish the program

Program length = 29 bytes
Checksum = 7D99

To get the period of the pendulum if it is 25 cm long:

1. Press SHIFT SOLVE and FN P to select the program P. (You do not need to do this if you have not selected any other program for use with Solve or Integrate since last selecting P.)

2. Press SHIFT SOLVE and SOLVE P to find a value of P which makes the expression equal to zero.

3. The HP-32S stops and displays L?0.2482 to show the present value of L and to ask if you want a new value. Type .25 and press R/S to solve for the period.

4. Wait a couple of seconds and see the answer 1.0035 - the clock will be 0.35% slow (about 5 minutes a day).

You can now use the same program to see what length you would require if you wanted a period of 2 seconds:

1. Press SHIFT SOLVE and SOLVE L to find a value of L - note that you do not need to select the function (program) P, because it is still selected.

2. See the HP-32S show P?1.0035 - type 2 R/S to solve for a period of 2 seconds.

3. Wait a short while and see L=0.9929, the length needed to give a period of 2 seconds.

We could make the value of "g" a variable too. If you wanted to calculate periods very exactly in different places then you could use INPUT G and RCL G instead of 9.8 in the program. This would let you calculate the periods of pendulums on the Moon, or Mars, or other exciting places too! You might have had enough of pendulums for now, but there is another important point to note. Each of these programs was more than 10 lines long. They are still short programs, but it is easy to make a mistake, even when typing a program this short. To help you check if the program is correct, the checksum of each program is given at the end. You should compare this with the checksum after you have typed in your program. To do this press SHIFT MEM, then PGM and use the down-arrow if necessary to move to LBL P. You will see the program length in bytes, and if you press SHIFT SHOW then you will see the checksum. If the HP-32S could be used with a printer, then I could print out each program, and you could compare your program with that printout,

or even make you own printout to compare. However the HP-32S does not use a printer (this helps make it less expensive!) so it provides checksums instead to help us compare our programs. This checksum is important, especially for longer programs.

2.3 BLACK BODIES AND BLACK HOLES In the previous example we started with a program to calculate a period given a length. This required only one calculation, so the program went through the formula once and gave the answer directly. After that we used the Solver to look for answers that satisfied specific conditions, and the program had to be used many times, until the Solver found a zero value. Using the Solver is a neat trick, since you only have to write the basic equation, and the Solver does the hard work of solving it. There are a few disadvantages though. Firstly, the Solver takes a while to find an answer - it has to guess instead of just getting the right answer directly. Secondly, it can sometimes guess a wrong answer. In the next example we shall put a little more effort into writing our own program, so that it does not need to use the Solver.

Calculating the properties of a black body requires a long program, but using those properties, along with quantum mechanics, gives a simple formula for the lifetime of a black hole.

$$L = M^3 * 10^{-31}$$

M is the mass of the black hole in kilograms, and L is its lifetime in seconds. This is the time before the black hole loses all its mass by pair creation at the Schwarzchild radius - and either vanishes or becomes a naked singularity, according to the theory you prefer!

We can write a program which begins by storing 0 in both L and M. Then it can ask for values of L and M. The user can type in a value for the known variable, and just press R/S to skip the unknown variable. The program can check which variable is still zero - and

can calculate that variable's value from the other one. Here is the program:

```
B01 LBL B     Label - Black hole.
B02 0         Get 0
B03 STO L     and store it in L and in M
B04 STO M     to show they have no value.
B05 INPUT L   Get L.
B06 x=0?      If it is zero then it was skipped,
B07 GTO C     so go to ask for M.
B08 31        Get 31,
B09 10^x      calculate 10^31
B10 *         and multiply.
B11 3         Now get 3.
B12 1/x       Calculate 1/3.
B13 y^x       Get third root.
B14 STO M     Store the result in M,
B15 VIEW M    and show it.
C01 LBL C     Come here to continue by asking for M.
C02 INPUT M   Ask for M.
C03 x=0?      If no value given for M or L,
C04 GTO B     then go back to ask again.
C05 3         Get 3.
C06 y^x       Get M cubed.
C07 31        Now get 31,
C08 10^x      calculate 10^31
C09 ÷         and divide.
C10 STO L     Store the result in L,
C11 VIEW L    and show it.
C12 RTN       End the program.
```

Program lengths: B = 22.5 bytes, C = 18 bytes
Checksums: B = E63A, C = 7C89

Type this into your HP-32S, check the program lengths and checksums, and correct the programs if necessary. Actually, it is all one program to calculate either the mass, or the lifetime, of a black

hole, but the HP-32S separates program memory into easily managed chunks by treating everything between one label and the next as a separate part. In this program there is no RTN before LBL C, so the program that begins at LBL B just carries on into the chunk labelled with a C. The part labelled B stops at the VIEW M step, but in a different program B could have just carried straight on into C.

Now try the program out. What would be the lifetime of a black hole with the mass of the Earth?

1. XEQ B

2. See L?0.0000 - press R/S to leave this unchanged, as we want to calculate a lifetime.

3. See M?0.0000 - type 6E24 - approximately the mass of the Earth in kg, then press R/S.

4. See L=2.1600E43 - a black hole the mass of the Earth would survive this many seconds, or about 7E35 years.

Assuming the age of the universe to be about 15 billion years, what is the original mass of the smallest black hole that would still be around if it was created at the big bang?

1. XEQ B

2. See L?0.0000 - calculate 15 billion years in seconds - 15E9 ENTER 365.25 * 24 * 3600 * and let the program run with this value of L by pressing R/S.

3. See M=1.6791E16. Black holes of about this mass or more would still exist (though their mass would be much lower by now, and if they were originally close to this mass then they would now be white-hot - not bad for a black hole!)

What do we learn from this? (Apart from something about the theory

of black holes, that is!) Instead of using Solve to look for an answer, we have written a program which decides which answer we want, and then calculates that answer at once. This takes some more program memory, and some more of the user's time, but it gives answers much more quickly. The same trick is worth using even in calculations which use more than two variables.

We have used two other "tricks". One is that of letting two programs run together, by not bothering to have a RTN before the second label. This is done very often in large programs, but it deserves to be emphasised even here. The second trick is that of using as little space as possible to store constants. The program uses 10^{31} in two places. Storing this number in a register would use 8 bytes, plus 3 more bytes in the two places where it is recalled - a total of 11 bytes. Getting the number by using 31 and 10^x uses 3 bytes (positive integers below 100 take up only 1.5 bytes), so doing it twice uses 6 bytes - a significant saving. People often use 31 CHS 10^x * without thinking, instead of using 31 10^x ÷ which saves one and a half bytes. If the program had used the actual number 1E31 then this would have taken up 9.5 bytes in each place - a total of 19 bytes, instead of the 6 actually used. That would indeed have been a black hole swallowing up valuable bytes!

2.4 MORE VARIABLES - THE IDEAL GAS LAW The ideal gas law is another example of a commonly used equation which can show some useful features of the HP-32S. (It is one of the equations built into the HP-22S.) The equation is:

$$P*V = N*R*T$$

This involves the constant R (R=8.3143 J/mole-K in SI units) and four variables. We could use the approach of section 2.2 above, and write a program which asks for all the variables, then solves for the unknown one, or we could use a program which uses the Solver to find the unknown value. Let us use the latter, to make the program as short as possible. The formula to be turned into a program is:

$$P*V - N*R*T = 0$$

```
G01 LBL G       Gas law program
G02 INPUT P     Get P, the pressure in N/m^2
G03 INPUT V     Get V, the volume in m^3
G04 INPUT N     Get N, the number of moles of the gas
G05 INPUT T     Get T, the temperature in degrees Kelvin
G06 RCL P       Now recall P,
G07 RCL* V      multiply by V,
G08 8.3143      put in the gas constant,
G09 RCL* N      multiply it by N,
G10 RCL* T      multiply by T,
G11 -           and finally subtract NRT from PV.
G12 RTN         End program - returns to the Solver.
```

Program length = 26 bytes
Program checksum =0696

Enter this program, see that your checksum is correct, then use the program to calculate the pressure of an ideal gas at temperature 273 degrees Kelvin, with 3 moles in a 4 liter container.

1. SHIFT SOLVE FN G - selects the gas program to be solved
2. SHIFT SOLVE SOLVE P - select P as the unknown variable
3. See V? - type 4E-3 R/S (4 liters is 4E-3 M^3)
4. See N? - type 3 R/S
5. See T? - type 273 R/S
6. See P=1,702,352.925 - the pressure in N/m^2 (or 16.8009 atm.)

Note: the value in N/m^2 is given to more significant figures than is meaningful, and you would not normally write an answer like this. The number is given here in the way it is displayed by the HP-32S if you are sticking to FIX 4 display mode as suggested at the beginning.

The program shown above does not introduce any stunning new tricks - it just shows how a formula with many variables is written for the

Solver. The use of RCL arithmetic makes the program considerably shorter than it would otherwise be. One trick is worth mentioning - the number 8.3143 takes 9.5 bytes in the program. If you were very short of memory you could save some bytes by working the number out using only positive 2-digit numbers. A way to do this is:

97 3 * 35 /

You can enter two numbers one after another, as shown above, by typing in the first number, then pressing - and backarrow (this makes sure the HP-32S knows that the first number is finished), then typing the second number. (HP-41 users note - there is no wasted space between the two numbers as there would be on an HP-41.) This uses five steps, each 1.5 bytes long - a total of 7.5 bytes instead of 9.5 bytes. While you are writing short programs and still have free memory left then this sort of thing is hardly worth the trouble, but when you are writing long programs it is worth remembering. You may think that even though this saves 2 bytes it wastes a lot of time - well, yes the program step 8.3143 takes 2.5 milliseconds, whereas the five steps shown above take 26 milliseconds - but that is still very little time!

2.5 DEFLECTION OF A CANTILEVER BEAM The equations used up to now could all be solved for any variable simply by having the variables rearranged. Let us now see how the Solver deals with a fairly simple problem which is not linear in all the variables.

The equation for the vertical displacement of a cantilever beam (a horizontal beam fixed at one end) with a load P applied at the free end is given by:

$$\frac{P*x^2*(3*l - x)}{6*E*I} - y = 0$$

l = length of beam
P = load on free end

E = Young's modulus for beam
I = Moment of inertia of beam cross-section
y = vertical displacement, at horizontal distance x from fixed beam end

Rewriting this equation to solve for x would be quite a nuisance, so let's see how the Solver will deal with the equation as it stands:

```
B01 LBL B       Beam equation
B02 INPUT P     Ask for P
B03 INPUT E     Ask for E
B04 INPUT I     Ask for I
B05 INPUT L     Ask for L
B06 INPUT X     Ask for X
B07 INPUT Y     Ask for Y
B08 RCL P       Get P
B09 RCL* X      Multiply by X
B10 RCL* X      And again - a quick way to get x^2
B11 6           Get 6
B12 ÷           and divide by it
B13 RCL÷ E      Divide by E
B14 RCL÷ I      and by I
B15 3           Get a 3
B16 RCL* L      Multiply by L
B17 RCL- X      Subtract x
B18 *           Now multiply by (3l-x)
B19 RCL- Y      Finally subtract y
B20 RTN         and finish the program
```

Program length = 30 bytes
Program checksum =C406

We'll use this program in English units - the formula applies for metric or English standards, as long as consistent units are used throughout the equation for the measurements and for Young's modulus. Assume a beam of 120 inches, with a moment of inertia of 5 inches^4 and a Young's modulus of 30E6 psi. If a load of 200 pounds

is placed on the free end, what will the deflection be at 100 inches away from the fixed point?

1. Type 0 STO Y (this is an initial guess for Y), then SHIFT SOLVE FN B

2. SHIFT SOLVE SOLVE Y

3. See P?1,702,352.925. This is the P value we got from the previous example - if you have cleared P or used it for something else since the previous example then P will contain something different. Type 200 R/S to give the P for this example.

4. See E?(something, it doesn't matter what). Type 30E6 (or 3E7) R/S.

5. See I? - type 5 R/S.

6. See L? - type 120

7. See X? - type 100 R/S.

8. You should see SOLVING in the display. If you have done all the examples so far and have not deleted any, then you might see MEMORY FULL - the HP-32S needs to grab 33.5 bytes to store numbers used during solving, and this much space might not be available. In that case you will have to delete some programs or variables - use SHIFT MEM to check how much memory you have left and what programs and variables you have - then use SHIFT CLEAR to delete some things you no longer need - the statistics area is a good place to begin, unless you still need some statistcs values you have previously entered. While checking memory you might find that there are more than 33.5 bytes free, yet you got MEMORY FULL - this happens if there is not enough memory for the Solver and for the unknown variable Y.

9. If the Solver ran out of memory, go back to step 2, but just

press R/S for all inputs, since you have already typed in all the values.

10. When the program does run successfully you should see that the beam displacement y is 0.5778 inches. You will also notice that this example takes longer than the previous ones to find a solution.

11. Now find the displacement at the end of the beam, where X is equal to L, 120 inches. Repeat the above, but press R/S at each input without entering a new value, except for X where you should enter 120.

12. The Solver takes a much shorter time now to find that the displacement at the end is 0.7680 inches.

13. Now go back and check the program by finding the X value which gives a displacement of 0.5778 inches. Repeat everything, but SOLVE for X, and give the value 0.5778 to Y. Even though there is a good initial guess for X left in X from the previous calculation, the Solver still takes a fairly long time to find the answer, which it gives as X=100.024 inches.

The last part of the example demonstrates two points. First of all, the Solver takes considerably longer to find an answer if the equation involves powers or functions of the variable being solved for - it is still fast, and certainly takes less time than it would for you to rewrite the equation. Secondly, the answer is not exactly 100 - the Solver finds a solution for X when Y is equal to exactly 0.5778 - the actual value of Y when X was 100 inches was 0.577777777777, and with this value in Y we do get a result of exactly 100 for X. This shows the effects of rounding - but we would be unlikely to need results more accurate than this anyway.

2.6 BODY SURFACE AREA Here is another example of a formula which involves powers and functions of variables. The body surface area (BSA) of a person, used particularly in cardiac medicine, can be

estimated from the height and weight of that person using the Boyd formula:

$$BSA = W^{(0.7285 - 0.0188 \log W)} * H^{0.3} * 3.207E\text{-}4$$

BSA = estimated body surface area in m^2
W = body weight in grams
T = height in cm
These are not SI units, but let's stick to the units of the original equation.

You could try to rewrite this formula to solve for height or weight, given the other two, but it is much simpler to let the Solver sort this out. In any case, the formula is nearly always used to find BSA. Let us write two programs, one of which calculates the right-hand side minus the left-hand side (so it can be used by the Solver), and a second program which calculates BSA directly from the formula, or uses the Solver to find either of the other two variables.

The program to find BSA is:

B01 LBL B	Label B, short for BSA
B02 RCL W	Get W
B03 0.7285	Get first part of power
B04 RCL W	Get W again
B05 LOG	LogW
B06 0.0188	Next constant
B07 *	0.0188 LogW
B08 -	0.7285 - 0.0188 LogW
B09 y^x	Now calculate $W^{(0.7285 - 0.0188 \log W)}$
B10 RCL H	Get H
B11 3	These three steps
B12 10	calculate 0.3 - put 10 after 3 by pressing ÷ <- 10
B13 ÷	in 4.5 bytes instead of using 9.5
B14 y^x	Get $H^{0.3}$
B15 *	multiply first two terms

```
B16 3.207E-4    Get last constant
B17 *           and multiply to give the formula
B18 RCL B       Get B
B19 -           and subtract, giving the final result
B20 RTN         Finish, or go back to calling program

Program length  =  54 bytes
Program checksum = D949
```

This program does not input any of the variables - we shall leave that up to the main program which "calls" this program to do the calculation. Note that if the variable B is zero then this calculates the predicted value of the BSA.

Now for the second program, which uses the above program to calculate any of the required variables.

```
M01 LBL M       Label M - Main part of program.
M02 0           Get 0,
M03 STO W       and store it in W,
M04 STO H       and in H,
M05 STO B       and B.
M06 SF 0        Set flag 0
M07 INPUT W     Input the weight.
M08 x≠0?        If not 0 then W is not an unknown,
M09 CF 0        so clear flag 0.
M10 INPUT H     Input the height.
M11 x≠0?        If not 0 then H is not an unknown,
M12 CF 0        so clear flag 0.
M13 INPUT B     Input the body area.
M14 x=0?        If it is zero,
M15 GTO D       then go to D to calculate B Directly.
M16 FS? 0       Is the flag still set?
M17 GTO E       If so then neither H nor W were given so show error
                message.
M18 FN= B       We have B and either H or W, so we must SOLVE
                function B.
```

M19 23	Put 23 (W) in stack.
M20 8	Put 8 (H) in stack - press + <-- 8 to put it after 23.
M21 RCL W	Get W.
M22 x=0?	If it is 0 then we must solve for W,
M23 R↓	so roll down the 8.
M24 R↓	Roll down once again, to get 8 or 23 in register X.
M25 STO i	Put 23 (W) or 8 (H) in the indirect register.
M26 STO (i)	Store this in unknown variable, otherwise first guess is 0.
M27 SOLVE (i)	Solve for the selected variable.
M28 VIEW (i)	Display the result.
E01 LBL E	Get here if SOLVE failed, or from an error at M17 or D04.
E02 10	To show an error message,
E03 ACOS	try to take ACOS(10) - this stops and shows INVALID DATA.
E04 GTO M	After an error, go back to M and start again.
D01 LBL D	Label D - direct calculation of B, without using SOLVE.
D02 RCL W	Get W.
D03 RCL* H	Get H and multiply.
D04 x=0?	If either is zero then product is zero,
D05 GTO E	which is wrong (both must be known), so go to error message.
D06 XEQ B	If neither is zero then calculate B.
D07 STO B	Store result in B.
D08 VIEW B	Display the result.
D09 RTN	Finish the program.

Program lengths M = 42 bytes, E = 6 bytes, D = 13.5 bytes
Program checksums M = B94A, E = 9663, D = E06C

Before going through an explanation, let's try the program out. First the simple case, what is the estimated body area of a person with height 170 cm and body weight 70 kg?

1. XEQ M (execute the Main program).

2. See W? - type in 70,000 (must be gm), and press R/S.

3. See H? - type in 170 (cm) and press R/S.

4. See B? - this is the unknown, so just press R/S.

5. See RUNNING and then the answer B=1.8347 at once. This person's body surface area is estimated at 1.8347 square meters.

Now try to find the the height predicted by this formula for a person with height 170 cm and BSA 1.8347 square meters.

1. XEQ M.

2. See W? - press R/S as this is the unknown.

3. See H? - press 170 and R/S.

4. See B? - press 1.8347 and R/S.

5. See SOLVING for several seconds as the Solver struggles to find an answer to the non-linear equation (it involves powers and logarithms), then the answer W=70,002.0749. Once again, the answer is not exact because the value of B was given to only 4 significant figures. This hardly matters, since the equation provides only an estimate.

If you try to find a solution but do not give two out of the three variables then the program cannot give an answer, and it might try to take the logarithm of zero as well. It therefore makes good sense to check for such an error and display an error message if necessary. The HP-32S does not let you write your own messages, but you can use one of those built into the HP-32S, as is done by the program with label E - see point E. below.

If you give negative values for the weight, height or body surface area, then the program will stop with some sort of arithmetic error, such as LOG(NEG). The program could check for these errors too, but this would take up more memory than is worth the trouble - a negative value is clearly a mistake!

If you give nonsense values then the program will work, but will give a nonsense answer! For example, remember to give the weight in grams. This is an example of the famous GIGO law of computing - Garbage In Garbage Out!

If you give values for all three variables then the program solves for H.

Now let us look at the tricks and special techniques used by the program.

A. First of all, the variable B contains 0 if it is the unknown variable, so we can use the program B to find B directly in this case, since it calculates the right-hand side of the original equation, then subtracts 0 from it. If B does not contain 0 then we are solving for one of the other variables anyway. This trick lets us use the main program to calculate B directly, without using the Solver. Note that LBL B and the variable B have been chosen to have the same name, but you can choose any names - LBL B and variable B are not the same thing.

B. The program uses flag 0 to check if at least one of the values H or W was given when B is given too. If flag 0 is still set then neither was given, so the error message is displayed. In this case, flag 0 stays set, but the program will presumably be used again, and if the program runs correctly then flag 0 is cleared - which is the normal default. It is usually best to leave the flags clear at the end of a program.

C. To solve for either H (variable number 8) or W (variable number 23), the program puts both of these numbers on the stack, then puts

W in the stack and checks if that is zero. If W is zero, then it is the variable to solve for, and the program rolls the stack down once to remove W, and again to remove the 6. If W is not zero, then H is the variable to solve for, so the program rolls down only once to remove W. Following this, the program stores the number in stack register X into the indirect register "i". The value stored in the unknown variable is still 0, and if this is used as the first guess by the Solver then it might produce a LOG(0) error, so the program stores the value in register X into the unknown variable to avoid that. We could actually put a more sensible guess into the unknown variable, but this trick saves some space and does fulfill the main purpose - which is to use a non-zero value as the first guess. Now the program solves for the variable given by "i", and displays this variable.

D. If for some reason there is no solution, then the step after SOLVE is skipped by the HP-32S, and the program falls directly into the next program - the error program labelled E. Since the part of the program beginning at label M does not finish with a RTN, the program can "fall" straight into the next one - in fact it is all the same program.

E. The "error message" program beginning at label E tries to calculate ACOS of 10, making the HP-32S stop and display INVALID DATA, a very appropriate message.

F. The part of the program that begins at D calculates B directly. First of all it must check that values have been given to both W and H - if neither has been given then it is zero. Instead of checking W and H separately, the program multiplies the two together and checks if the product is zero - thus saving 3 bytes. If neither W nor H is zero then the program uses program B to calculate the value of B, then it stores the result in B and finishes by displaying this value. At this point the program does finish with a RTN to make sure that an accidental push on R/S does not start another program. This also means that yet another program could use M, and when M was finished it would return to that program.

2.7 SAVE SPACE AND TIME Well, this chapter is supposed to be about "short programs", and you might feel that the program above which begins at label M is not exactly short. It is certainly the longest program so far in this book, but we shall see a longer one in the next chapter. Let us finish this chapter with a summary of tips on saving time and memory. (Yes, that's what this section is about - it is about saving memory space and program time in your HP-32S - it is not a plea from a relativistic environmentalist.) The tricks used in the program above showed several examples of that. If you are short of memory then delete the program written above unless you plan to use it again.

That is one of the memory-saving tricks you should remember when using the HP-32S - delete unwanted programs as soon as you have finished with them. Otherwise you might forget what the program does and be scared to delete it in case it is important. If a program is important but you no longer need it then you should copy it into your notebook (mentioned at the beginning of Chapter 1). Write down its checksum too so you can check the program if you ever type it in again, then delete it at once.

A similar tip is that you should delete variables as soon as you no longer need them. A particularly neat way to do this is:

RCL v STO- v (Where "v" is any variable)

This recalls the variable, then subtracts it from itself, leaving 0 in the variable, and thus saving the space it would otherwise occupy.

Mention of STO- brings us to the whole range of STO and RCL arithmetic. Use these wherever you can - using them makes programs shorter, and therefore faster too. For example if you need to calculate 1.7*Q then you could do:

RCL Q 1.7 *

but the following is faster:

1.7 RCL* Q

If you are really short of memory then using the number 1.7 in a program takes an unnecessarily large amount of space (9.5 bytes). The following is shorter:

17 10 ÷ RCL* Q

- it takes 4.5 bytes instead of 9.5 bytes for the number - but of course it is a bit slower. A similar trick is to store small negative numbers as:

number CHS instead of -number

This brings us to another point about numbers in programs. Whenever you begin to type a number into a program the HP-32S checks if you have at least 9.5 bytes of memory free to hold the number. This happens even if the number you are typing will actually take only 1.5 bytes. After all, when you begin to type a number, the HP-32S does not know if you plan to type in a zero or a positive integer below 100, and only those numbers take 1.5 bytes. Say you have just 8 bytes of memory left, and you want to finish a program by typing:

35 + 3 y^x

That would take only 6 of your 8 remaining bytes, but the HP-32S will say MEMORY FULL as soon as you type the first "3". You can still get this piece of the program in - go back and delete three previous steps of the program (not numbers though!), so you have 12.5 bytes left, then type in the 35 and +. You now have 9.5 bytes left; exactly enough to type in the 3 - after which you can put in the last step, then go back and put in the steps you have had to delete. If you are typing in a long program it may well be worth typing in all the short numbers first, then going back and typing in the rest of the program, so you will not be in danger of having to

delete and re-enter some steps at the end.

Apart from using short numbers whenever possible in programs, you can save memory by avoiding unnecessary instructions. A RTN at the end of a program is useful to show that the program is finished, and if you press R/S by accident then it stops you again (but see below). If you are very short of space though, then these are unnecessary luxuries! A program which stops with a VIEW or a STOP does not really need a RTN as well, and the last program in memory does not need a RTN either, since the program stops when it comes to the end of memory. If you are really short of memory and a program sets modes with commands such as DEG or SCI 7 then you can remove these steps and set the modes from the keyboard before using the program.

There are times when you cannot remove RTN as described above. RTN is the "return" instruction, and when one program uses XEQ to run another one as a subroutine the RTN is used at the end of the subroutine to go back to the first program. In such cases, the RTN is necessary, unless the subroutine is the last program in memory (just before PRGM TOP) - in which case the end of memory behaves like a RTN.

Well, having reached the end of memory, we have also reached the end of this chapter. The next one will use two fairly long programs to show further tricks worth using in programs.

CHAPTER 3 - LONGER PROGRAMS

3.1 STATISTICS AND LOOPING In this chapter we shall study two programs which add extra features to the HP-32S. They will use some of the tips and tricks introduced earlier, and will show a few new useful ideas. In effect, these examples are "case studies" of advanced programming on the HP-32S. Our first example will be a program to help in the use of the statistics functions.

There are three possible problems with the statistics functions built into the HP-32S. One is that you cannot keep track of each number used - they are just added up into the summation registers. If you get an unexpected mean value and want to review the numbers you typed in, to see if one was wrong, you cannot recall each number you had entered. Some of the more expensive HP calculators keep a "list" of all the numbers used, and the program below will make a list like that as well, provided you do not use more numbers than the HP-32S can store. The second problem is that large numbers with small variations can give inaccurate statistical results. This can lead to the third problem, small differences between large numbers can cause an underflow and then standard deviations cannot be calculated at all. The program below calculates a mean value, then subtracts it from the list of measurements, and recalculates the mean and standard deviation more accurately. It also lets you review the numbers you entered, in case some need to be corrected.

S01 LBL S	Statistics program
S02 CLΣ	Clear out any old statistics values
S03 0	
S04 STO i	Set loop counter to 0
E01 LBL E	Loop to Enter statistics values
E02 STOP	Stop, show number of values so far, wait for next value
E03 ISG i	Increase counter to store next x value
E04 RADIX.	A do-nothing instruction to follow ISG
E05 STO (i)	Store x value

E06 x<>y	Swap x and y values
E07 ISG i	Increase counter to store next y value
E08 RADIX.	Do-nothing instruction again
E09 STO (i)	Store y value
E10 x<>y	Swap back x and y values
E11 +	Add this pair to the statistics registers
E12 GTO E	Go to get next value or pair of values
M01 LBL M	Show the Mean values and standard deviations
M02 $\overline{x}$	Get mean x
M03 STOP	Stop to display it
M04 $\overline{y}$	Get mean y
M05 STOP	Stop to display it
M06 sx	Get standard deviation on x
M07 STOP	Stop to display it
M08 sy	Get standard deviation on y
M09 STOP	Stop to display it
R01 LBL R	Continue into Review section
R02 XEQ T	Count total number of values to review and put in i
C01 LBL C	Loop to Correct any values
C02 RCL(i)	Get original x
C03 ISG i	Increase counter so as to...
C04 RCL(i)	get original y
C05 x<>y	Swap x and y
C06 -	Subtract the original values from the statistics registers
C07 DSE i	Decrease counter again to get at original x
C08 RADIX.	Do-nothing step, otherwise DSE would skip the next step
C09 INPUT(i)	Display present x value and let user change it
C10 ISG i	Increase loop counter
C11 INPUT(i)	Display present y and let user change it
C12 DSE i	Decrease counter again...
C13 RADIX.	and put in a do-nothing step
C14 RCL(i)	Recall new x in case stack was changed by calculation of y
C15 +	Add new x and y (they might not have changed!) to stats

C16 ISG i	Increase counter back to position of y
C17 ISG i	Increase counter to position of next x if any more left
C18 GTO C	Go back to continue Correction loop if any more values left
C19 GTO M	If all values done then GTO C is skipped - go back to M
N01 LBL N	Normalize the stored values by subtracting the current means
N02 XEQ T	Count total number of values to review and put in i
N03 $\bar{x}$	Get mean x
N04 +/-	Change its sign, so it can be subtracted
N05 $\bar{y}$	Get mean y
N06 +/-	Change its sign, so it can be subtracted
N07 CL	Clear stats registers to store the normalized set
L01 LBL L	Loop to normalize
L02 RCL(i)	Recall next x value
L03 ISG i	Increase counter to get next y value
L04 RCL(i)	Recall next y value
L05 CMPLX+	This subtracts mean x and y from next x and y in one step
L06 STO(i)	Put back normalized y
L07 x<>y	Swap x and y
L08 DSE i	Decrease counter to get position of x
L09 STO(i)	Put back normalized x
L10 +	Add to new statistics set
L11 R	Remove x,
L12 R	and y, putting mean x and y back in Y and X registers
L13 ISG i	Increase counter again (back to y position), no skip
L14 ISG i	Increase i to next x, will skip next step if all done
L15 GTO L	Repeat loop if any more x,y pairs left
L16 GTO M	If all done then go to display new means
T01 LBL T	Subprogram to get Total number of values to review
T02 n	Recall number of values
T03 2	
T04 *	Double it (as there are x and y values)
T05 3	

T06 10^x	Calculate 1000 in 3 bytes, (save 6.5 bytes)
T07 ÷	Divide number of values to give loop limit
T08 1	
T09 +	Add 1 to give initial register (A) for loop
T10 STO i	Store this loop counter in i
T11 RTN	Return to place which called T (also marks program end)

Lengths S= 6 , E= 18 , M=13.5, R= 3 , C=28.5, N=10.5,
Checksums S=412A, E=D981,M=C3CE, R=EFC4, C=68E6, N=127C,

Lengths L = 24 , T=16.5
Checksums L =94E1, T=22A1

Most of what this program does is explained in the comments next to each line - let us have a look at a short example before looking at the special tricks used.

An experiment gives four readings which are 1,875,015.3
1,875,017.1
1,875,016.8
and 1,875,014.9

Find their mean and standard deviation.

1. XEQ S to run the statistics program. See 0.0000 - 0 results have been stored so far.

2. Type 1875015.3 and press R/S. See 1.0000 - 1 result has been stored.

3. Type 1875017.1 and press R/S. See 2.0000.

4. Type 1875016.8 and press R/S. See 3.0000.

5. Type 187514.9 and press R/S. Yes, I know this is wrong, but we are going to use it as an example of something going wrong. See

4.0000 - we have now got four results stored, and that is everything we have, so we can now look at the mean value and the standard deviation.

6. XEQ M - to look at the mean values. See 1,453,141.0250 - you notice that something is wrong because the value does not begin with the digits 1,875 as all the results did. This will tell you that at least one result was entered wrongly (or that something is wrong with the program?). Press R/S and see 1.5000 - this is the mean of the y values, but we are not interested in it. If the measurements had consisted of x and y values then we would have typed in pairs of values, separated by ENTER, in points 2 to 5 above, and we would be interested in the y mean.

7. As we are not interested in the y mean, press R/S again at once and see 843,750.7500 which is the standard deviation on the results. This is clearly wrong, since all the results were within a few units of each other - we shall have to review the values we typed in. Press R/S and see 1.2910 which is the standard deviation on the y values, which we are not using.

8. Press R/S again to go on to the review section. We could do XEQ R instead. See A?1,875,015.3000. This is the first result and is correct. Press R/S and see the value of B - the first y value, which we are ignoring, so press R/S again. If we had been using x and y values then we would have wanted to check the y value too.

9. See the correct value 1,875,017.1000 in C, so keep pressing R/S to ignore y values and check x values. At G?187,514.9000 we see that the number is less than a million - it must be the wrong one. Type in the correct value 1875014.9 and press R/S. See the value of H which is to be ignored again, so press R/S again. Since the reviewing procedure subtracts the old values from the statistics registers and then puts in the new ones, it is dangerous to interrupt the review procedure at a random place. It is best to go through the whole process - this also makes sure you find any further mistakes, instead of stopping at the first mistake and

assuming there are no more!

10. Press R/S again (or XEQ M) to see the new means and standard deviations. The mean is now 1,875,016.025 but the standard deviation gives STAT ERROR ! This can happen when the numbers are all large, particularly if the differences between them are small. It is one reason for using the next part of the program which "normalizes" the values to a range of values around zero, so the differences are not small compared to the sizes of the numbers. Since we cannot get the standard deviation, we shall go straight on to the normalization. That will change the mean values to zero, so be sure to make a note of the mean values first.

11. XEQ N - execute the normalization. When the normalization is finished it goes back to the display of the mean values. The new x mean is 0.0000, since the program has subtracted the mean value from each result. This confirms that the program has run correctly. Press R/S again to see the new mean value of the y values, which can be ignored (but should be zero as well), then press R/S again and see the standard deviation on the x values, which now comes out as 1.0874. A final press on R/S will show the standard deviation on y (meaningless in this example).

12. Well, that's it. By normalizing the original results we have made it possible to calculate an exact standard deviation. The results stored in the HP-32S have now all had the original mean value subtracted from them - if you need the original values you will have to add the mean to them again.

13. In this example I have provided very simple test results on purpose, so you can check if the above results are correct. It would have been just as easy to ignore the leading digits which were the same in all the results and to find the mean and standard deviation of 15.3, 17.1, 16.8, and 14.9. These would have been 16.0250 and 1.0874, as we found by using the program. That shows the program worked correctly!

Now we can look at the use of the program in general, and at the extra tips it shows. The program can be used to analyze up to 13 single measurements or x,y pairs of measurements. It is then using all of registers A to Z, and i, and the statistics registers; in addition, all except 6 bytes of the remaining memory are used to hold the program. Maybe you can think of 4 useful extra steps to put in those 6 bytes!

The loop at LBL E lets you put in pairs of numbers into registers 1 and 2 (A and B), then 3 and 4 (C and D), and so on. The loop uses a counter in the "indirect" variable "i". This counter is set at zero to begin with, but no upper limit is given - that means you can store away as many results as your HP-32S memory will allow. If you have only this program in memory then you can store up to 13 pairs of measurements in registers A to Z - if you try to store any more you get the message INVALID (i). If you have some other programs in memory then you will be able to store fewer results - it would be wise to do CLVARS (SHIFT CLEAR VARS) before using the program in this case, so as to make sure you can use as many variables as possible. If you use this program with other programs in the HP-32S then you can keep storing measurements until you see the message MEMORY FULL. Most likely you will not get either message but will simply put in all your variables (it is surprising how many lab experiments consist of taking exactly ten measurements!). No matter how you finish - with all measurements stored or with an error message, you can carry on to find the means by using XEQ M.

The program uses ISG to repeatedly add 1 to the counter - this is simpler than using 1 STO+ i R↓. Since the program imposes no upper limit on the number of measurements, the loop counter begins with 0.0000 - this means that the upper limit for ISG is zero and ISG will always skip the next step after it. (The ISG is being used only to add 1 to i, not as a loop test.) As this step will always be skipped, you can put any instruction you like in there - but (just in case something goes wrong!) it is best to use an instruction which does nothing. Sometimes when you ISG or DSE to increase or decrease a number you will not know whether the step

after ISG or DSE will be carried out or not. Then you must make the step after ISG or DSE a do-nothing step - so it does not matter whether it is carried out or not. Many computers and calculators have a special instruction called a NOP (Null OPeration) for this purpose. The HP-32S has no special NOP command - I have used RADIX. because this does not alter the way results are calculated (some people use DEG as a NOP), nor does it change the way results are displayed. (If you use European notation you should put RADIX, in place of RADIX.) The only other instruction that it would be safe to use as a NOP would be a LBL, but there is only a limited number of LBLs, whereas you can use RADIX. as often as you like. This is an important tip - if you use ISG or DSE to increase or decrease numbers without using the skip part then use RADIX. after these instructions.

The program lets you work with pairs of results (measurements), or with single results. If you are using pairs of measurements then you should type in the y value, press ENTER, type in the x value, and press R/S. If you are using single measurements then press R/S each time. If you do this then the y value will normally be the number of measurements stored so far - this is the number in X at the beginning of the loop. If you use the stack to calculate each measurement before entering it, then the number in stack register Y might be almost anything - and it might make the y mean and standard deviation act oddly. If you do use the stack to calculate an x value then it is best to press ENTER R/S so that the x and the y values will be the same (and presumably safe). This is another tip - do not do statistics on the x register with undetermined values in the y register.

When you use XEQ M to see the mean values you are shown the x mean; then you press R/S to see the y mean, then R/S again to see the x standard deviation, and then the y standard deviation. (These are "sample" standard deviations - a footnote in Chapter 11 of the manual shows a trick to let you calculate "true population" standard deviations.) If you press R/S one more time then the program goes straight into the review section - another example of letting one

program section go straight into another without a RTN between them.

The review section lets you see each result - this is the difference between using this program and just using the statistics functions on their own. Remember that the results are in pairs (in the order x,y) - A then B for the first pair, then C and D, and so on. If you have been entering single measurements then the second value of each pair is irrelevant - just press R/S to skip it. If any value is wrong then you can just type in the correct value and press R/S - if a value is correct then just press R/S to see the next one. This time you want to check only the values you previously entered, so the loop counter "i" has to be stored with a limit. Since the counter is used again in the normalization section, it is set up by a separate "subprogram" or "subroutine" beginning with LBL T and ending with a RTN. This subprogram can be used by other parts of the program which just XEQ T to do the job. The subprogram ends with a RTN, and this RTN is also used to mark the end of the whole program. As I mentioned earlier, this RTN is not really needed if it comes at the end of the last program in memory. You can delete it from this program if you need to squeeze in five more program steps.

Since the review section does not know in advance which values (if any) will be corrected, it retrieves the x and y values of each pair, subtracts every measurement from the statistics registers before letting you change it, then adds it to them again. In order to do these manipulations, then let you change the values if desired, the program uses ISG i and DSE i alternately. The ISG steps do not skip the following steps until all the results have been checked, but the DSE steps do skip the following step, so RADIX. is again used as a do-nothing filler after DSE. To make sure that the x and y values are viewed in the expected order the program has to swap them from time to time. After each pair has been subtracted, checked, and added again, ISG is used twice - once to go to the x value, and again to get to the next pair. It is only this second ISG that can skip the following step. Since the measurements come in pairs, there is no need for a NOP after the first ISG of the

pair - the GTO C after the second ISG goes back to continue the loop until all the results have been checked. When the check has been accomplished the program goes back to LBL M to let you see the new mean values and standard deviations.

Once you are happy with the mean values, you can use the "normalization" section to subtract the mean x and y values from each measurement. As discussed earlier this technique brings the measurements to a range near zero, and thus allows the calculation of more exact standard deviations. Subtracting the x and y means from each measurement would require that they be stored somewhere, or that they be recalled from the statistics registers at each step. The new statistics results have to be worked out in a separate loop. To simplify all this, the negative x and y values are stored in the stack, and then subtracted simultaneously from each x and y pair by adding them to the pair with CMPLX+. This is a neat trick, and it lets you keep the mean values in the stack at all times so they do not use any of the variable registers. Before looking at the rest of the program let us have a quick look at how the complex stack works.

3.2 COMPLEX ARITHMETIC AND THE COMPLEX STACK The CMPLX+ trick described above works because of a special feature of complex arithmetic on the HP-32S. Normally when you use the stack, the addition operation drops the stack one value, and register T is duplicated. For example, the stack and the LASTx register might look like before and after an addition:

before +	after +
T 4	T 4
Z 3	Z 4
Y 2	Y 3
X 1	X 3
L ?	L 1

The previous X is saved in L, the stack values Z and T drop down, and a new copy of T is put in T. Now, if you use CMPLX+ to add two complex numbers:

$$z1 = a + ib \qquad\qquad z2 = c + id$$

the stack will look like this before and after the addition:

	before CMPLX+	after CMPLX+
T	d	d
Z	c	c
Y	b	b+d
X	a	a+c
L	?	a

Only the real part of z1 is saved in L, but the whole of z2 is copied into Z and T. The complex number at the top of the stack is copied when the HP-32S does complex arithmetic, just as the top real number is copied in real arithmetic.

This means that arithmetic operations with two complex numbers do save the whole of one of them - not in LASTx but in registers Z and T. Say you want to calculate z1 + 2*z2. You can not use LASTx to recover z1, as you would do with real numbers but you can swap the two complex numbers. Enter z1 first, or enter z2 and then z1 as usual, then press R↓ twice to swap the two complex numbers. Now you can just press CMPLX+ twice and get z1 + 2*z2. You can use this method in general to treat stack registers Z and T as a complex LAST.

The statistics program uses this, but with an extra twist. We need to subtract the mean x and y values from each pair, but with the least effort possible. The trick is to store the negative of each mean in registers Z and T, so the CMPLX+ operation does the

following:

before CMPLX+	after CMPLX+
T $-\overline{y}$	T $-\overline{y}$
Z $-\overline{x}$	Z $-\overline{x}$
Y y	Y $y-\overline{y}$
X x	X $x-\overline{x}$
L ?	L x

Neat, isn't it? Now, I'll let you into a secret - it would actually have been just as easy not to change the signs of the mean x and y values but to use CMPLX- CMPL+/- instead! This would have subtracted the values from the means, then changed the signs in both X and Y, giving the same result - but then we wouldn't have had a chance to discuss the above trick, would we? Now let's go back to the rest of the statistics program.

3.3 IMPROVED STANDARD DEVIATIONS After the mean values have been subtracted from all the values, and the new statistics values have been set up, the program goes back to LBL M again to display the new mean values and standard deviations. The new mean values might not be exactly zero if the statistics summations required more than 12 digits to store exactly. The new standard deviations should be the same as they were before - except if the original data used were large then the new standard deviations are more exact. The calculation of the standard deviation involves sums and differences of squares - and this is bound to lose accuracy if you are using large numbers. As was shown in the example, the original calculation can even overflow completely and give no useful result.

It is possible to obtain better accuracy by using an alternative set of formulae to store the statistical values. Articles on this have been published in mathematical journals, and in the magazines of HP user clubs. If you dislike the program presented here you can try

looking up the alternative formulae and using them instead.

3.4 INTEGRATION WITH INFINITE LIMITS The HP-32S integration function integrates between finite limits. One of the examples in the manual shows what can happen if you try to integrate to an infinite limit by using a very large number as the upper limit. If you want to integrate with one or two infinite limits you can do one of two things:

1. Write your own integration program which can integrate to finite or infinite limits. The High-Level Math Solution book for HP-41 users shows such a program which uses Gaussian quadrature. You might like to adapt that program for use on the HP-32S, or write a similar one of your own.

There is a second reason for writing your own integration program even if you do not need to work with infinite limits; the HP-32S Solver and Integrator cannot be used at the same time. If you want to Solve a function whose program uses integration then the HP-32S will not let you do that - nor can you integrate a function which uses the Solver. If you want to do either of these things then you <u>can</u> do it by writing your own integration program and using that instead of the built-in Integrate function.

2. You can change the variable of integration to a new variable whose range is finite. This is what we shall do here.

Whichever method you use, it is worth noting that the program need only integrate to one infinite limit:

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

Any program which does this also lets you integrate between minus infinity and infinity by doing the integration in two parts:

$$I = \int_{-\infty}^{\infty} f(x)\,dx = \int_{-\infty}^{0} f(x)\,dx + \int_{0}^{\infty} f(x)\,dx$$

$$= \int_{0}^{\infty} f(-x)\,dx + \int_{0}^{\infty} f(x)\,dx$$

In general $$\int_{-\infty}^{a} f(x)\,dx = \int_{-a}^{\infty} f(-x)\,dx$$

so a program for integration from a finite value to plus infinity can also be used to integrate from minus infinity to a finite value, or from minus infinity to plus infinity as shown above. A Gaussian quadratic integration program will usually use a selected number of intervals, so if you want to use more steps to obtain greater accuracy, you have to try something else. Replacing infinity with the largest number the HP-32S can handle can lead to nonsensical results since most of the values of the integrand will be calculated at very large values of x, as is shown in the manual. Looking for a reasonable upper limit by trial and error can take a very long time.

Let us therefore look at the alternative method of changing the variable that has infinite limits to one that has finite limits. The built-in integration program uses a user-supplied routine to calculate the function f(x) at a value x and then makes a choice of x values which lets it estimate the integral. If the variable is changed from one that has infinite limits to one that has finite limits then this general-purpose program can still be used provided that the routine to calculate f(x) is supplemented by a second routine that makes the change of variable. In the following, I shall call these routines F and T respectively. A reasonable choice for the change of variable is:

$$x = TAN(\Theta)$$

or

$$\Theta = ATAN(x)$$

because TAN(infinity) is pi/2 radians, so infinity can be replaced by pi/2, and the integral can then be rewritten:

$$I = \int_a^b f(x)\,dx = \int_{\arctan a}^{\arctan b} \frac{f[\tan(\Theta)]}{\cos^2\Theta}\,d\Theta$$

When you make this change of variable the following things happen:-

Where a lower limit is $-\infty$ it becomes $-\pi/2$

Where an upper limit is $+\infty$ it becomes $+\pi/2$

Where the routine to evaluate f(x) was F, it now becomes a new routine T which uses F as follows:-

```
T01 LBL T
T02 RAD
T03 RCL V
T04 ENTER
T05 COS
T06 x^2
T07 X=0?
T08 RTN
T09 STO n
T10 x<>y
T11 TAN
T12 STO V
T13 XEQ F
T14 RCL n
```

T15 STO- n
T16 ÷
T17 RTN

The steps do the following:

02 Put the HP-32S into RADians mode for correct integration.

03 Recall Θ from V (the variable we are integrating over)

04,05,06 Get $\cos^2\Theta$ but keep a copy of Θ in register Y

07 & 08 If $\cos^2\Theta$ is zero, then return with zero in stack register X since the integrand has to be zero at pi/2 or -pi/2 if it is a closed integral. $\cos^2\Theta$ is compared to zero in preference to Cos Θ as it reaches zero sooner.

09 Store $\cos^2\Theta$ in a variable. You can use any variable you like so long as it is not used by the function program F, nor by any other program which uses the INTEGRATE command to work out the integral.

10 & 11 Put tan Θ into variable V. (It would be infinite if cos Θ were zero, but the RTN at line 08 avoids this.) Use the value of Θ which was saved in stack register Y. Tan Θ is equal to X, the original unknown variable used by the original function F before the variable was transformed. Putting the transformed value Θ and then X both in variable V may seem a little confusing, but it saves one register.

12,13 Now execute the original function F, or whatever its label is. Note that the angular mode is RAD; F may need to change it. We are assuming that F is a function of the independent variable stored in V, so we take the variable Θ from V, transform the variable using the program T, then pass the transformed value X to F.

14,15,16 Divide the result by $\cos^2\Theta$ and use STO- to put 0 in variable n so as to save space.

17 Return to the INTEGRATE command.

To calculate a numerical integral with an infinite limit using the HP-32S INTEGRATE command with T do:

1) Enter the function f(x) as a program in the HP-32S. The program is assumed to start with x in variable V and to return with f(x) in stack register X.

2) Enter the function T as above. At lines 09, 14 and 15 use a variable that is not needed by the integration or the f(x) program. At line 13, use the name of the f(x) program, or use the i register with the program label in it.

3) Run the HP-32S integration command as usual except:
   i/ Give the lower limit as ATAN(a) in radians instead of a, or as -pi/2 instead of - infinity.
   ii/ Give the upper limit as ATAN(b) in radians instead of b, or as pi/2 instead of + infinity.
   iii/ Give the function name T instead of the original function name (which I have called F in these notes).

For example calculate:

$$I = \int_0^\infty xe^{-x}\,dx$$

by doing the following:

1) Enter into your HP-32S the two programs given below

```
T01 LBL T
T02 RAD
T03 RCL V
T04 ENTER
T05 COS
T06 x^2
T07 X=0?
T08 RTN
T09 STO A
T10 x<>y
T11 TAN
T12 STO V
T13 XEQ F
T14 RCL A
T15 STO- A
T16 ÷
T17 RTN
```

```
F01 LBL F
F02 RCL V
F03 +/-
F04 e^x
F04 RCL* X
F05 RTN
```

Program lengths T=25.5, F= 9 bytes
Checksums T=A9A6, F=92B6
(These checksums will agree with yours only if you use A for the variable called n earlier on, and if you use the program names T and V, and the variable V.)

2) Do SHIFT SOLVE FN T to tell the HP-32S you want to integrate T.

3) Execute 0 PI 2 ÷ to give the lower and upper limits to the integration program. These are ATAN(0), which is itself 0, for the lower limit, and pi/2 to replace infinity for the upper limit.

4) Do SHIFT SOLVE then press the integrate FN key and press X to integrate over the independent variable X.

5) Wait for the answer: 1.0000 accurate to four decimal places.

For comparison, an attempt to estimate the integral of xe^{-x} from 0 to infinity by using the HP-32S integration function with the limits 0 and 1E499 gives the result 0.

The program T sets RAD mode - if you normally use DEG or GRAD mode then you must remember to set that mode after the integration is finished.

Naturally enough this whole section assumes that the functions to be integrated do not have any singularities in the range of integration and that the integrals are closed (finite).

This program does not show many new tricks for the HP-32S itself. The program T sets RAD mode every time, in case the mode is changed by F - this is a sensible precaution. RTN is used in two places - steps T08 and T17 - this shows that RTN does not have to be used only at the end of a program - it can be used wherever a subprogram needs to return to a program which called it. RCL n followed by STO- n is used to save space because this pair of instructions puts a zero into variable n, and saves 8 bytes. The two extra instructions (STO- n and ÷ which could otherwise be input RCL÷ n) use 3 bytes, so 5 bytes are saved.

The main purpose of this last program is to show an equally important trick - working things out for yourself. If the HP-32S does not do exactly what you want and you cannot find some clever trick using HP-32S functions then you should think through the problem and see if you can find a way to restate the problem in such a way that the HP-32S <u>can</u> deal with it. In this case we used changing the variable of integration as an example of that, but this is the most general of all tricks described in this book. It goes something like this -

THINK FOR YOURSELF!

Once you have mastered that final trick, you will truly be an expert HP-32S user. Maybe you will start finding new tricks and sharing them with others. Let us finish this chapter on that optimistic look into the future!

CHAPTER 4 - MORE INFORMATION

4.1 BOOKS AND SHOPS This (short) Chapter lists sources of further information for HP-32S use, including companies and clubs that can provide such information. First of all, the HP-32S comes with an Owner's Manual - I have suggested several times already that you should read it! A booklet of programming examples has also been published by HP, and more may be published.

Some of the HP-32S mathematical functions are taken from the HP-15C. The "HP-15C Advanced Functions Handbook" has many of the details that an advanced HP-32S user may want but cannot find in the Manual. It has the best discussion of numerical accuracy on calculators that I have ever seen. It is to be hoped that HP will produce a similar handbook for the HP-32S, in the meantime serious users should consider buying the HP-15C version.

Books of tips and suggestions for other calculators can give you useful ideas, in particular books for the HP-41. Probably the best book of this kind is "Calculator Tips and Routines - Especially for the HP-41C/CV" edited by John Dearing (1981) published by Corvallis Software Inc., P.O. Box 1412, Corvallis, OR 97339-1412, USA. Some of the tips in this book have come from ideas in HP club journals, see below. The Boyd formula for body surface area comes from Edith Boyd, "Growth of the Surface Area of the Human Body", University of Minnesota Press, 1935. It was quoted in the Cardiac/Pulmonary book of HP-41 Users' Library Solutions.

General RPN books are very useful too, in particular "Algorithms for RPN Calculators" by John A. Ball (1978) published by John Wiley & Sons, Inc. The book "Scientific Analysis on the Pocket Calculator" by Jon M. Smith (1977) is also published by Wiley - it gives detailed instructions for some important operations, it compares algebraic and RPN calculators, and has appendices with useful tricks

and operations such as matrix manipulations. If you get this book, look at its cover upside-down! A list of other books about the HP-41 and RPN calculators in general is given in Appendix A of my book "Extend your HP-41" (1985) also published by SYNTHETIX. Their address is P.O. Box 1080, Berkeley, CA 94701-1080, USA.

Books like these can be ordered for you by shops and bookstores which specialize in HP calculators. If there are none near you then try EduCALC Mail Store, 27953 Cabot Road, Laguna Niguel, CA 92677, U.S.A. who sell all these books and print a catalog of products, over 100 pages long, several times a year. They accept credit card orders from the U.S.A. or overseas by telephone and will send you their catalog if you ask for one. EduCALC also sells spare manuals for HP calculators. This is handy if you find a particularly good program for another HP calculator and want to translate it for the HP-32S, then you can buy a manual for the other calculator. Hewlett Packard also sells spare manuals for their calculators.

4.2 ELECTRONIC BULLETIN BOARDS If you have access to electronic mail then you will be able to access electronic bulletin boards. Some of these mention HP calculators in passing, and some have been set up by keen users of HP handheld calculators and computers. Several of these are available on FidoNet. There is also an electronic newsgroup called COMP.SYS.HP where people exchange ideas specifically on HP products. How you get access to these bulletin boards depends on where you are and what equipment you use for email. One way to get this information is from a user club.

4.3 JOIN A CLUB? Further information, on bulletin boards, and on real person-to-person meetings, is best obtained from user clubs. Smaller, local, clubs usually hold meetings once a month, where you can ask for advice or exchange ideas and information. Larger clubs including the U.S. club HPX publish regular journals as well - this is particularly useful if you live far from the nearest club meeting place. The larger clubs should be able to tell you if a local group

is active near you. Many user clubs run on the active work of only a few members, so their address and meeting place can change when a few members leave or join. When writing to user clubs, include a return address on your letter and if you write to one club and receive no reply, try another one - all will understand a letter in English.

UK and other European Countries

HPCC,
Geggs Lodge
Hempton Road
Deddington
Oxon OX5 4QG
United Kingdom

HP-GC
Quellinjnstraat 47-3
1072 XP Amsterdam
The Netherlands

PCX
Postbus 205
B-8000 Brugge 1
Belgium

PPC-Denmark
c/o Steen Peterson
Gl. Landevej 19
DK-2620 Albertslund
Denmark

PPC-Paris
BP 604
75028 Paris Cedex 01
France

CCD e.V.
P.O. Box 110411
Schwalbacherstr. 50 Hhs.
D-6000 Frankfurt 1
West Germany

STAK
c/o Tapani Tarvainen
Yliopistonkatu 10 B 21
SF-40100 Jyvaskyla
Finland

CHHU-IT
c/o Stefano Tendon
Cantone delle Asse 5
29100 Piacenza
Italy

Australia

PPPM Inc.
P.O. Box 512
Ringwood, Vic. 3134
Australia

CHHU Sydney
C/- K. Besley
Charlie Business Services
22 Elsie Street
Burwood, N.S.W. 2134
Australia

USA/International

HPX
P.O. Box 4160
Des Plaines, IL 60016
U.S.A.

This last club is located in the USA but acts as an international club too. HPX (The Handheld Programming Exchange) has taken over many of the activities of the club CHHU (which you may have heard of, but which has closed down, although local CHHU groups are active in some places.)

You may want to put new club addresses here.

INDEX

Other Products Available from SYNTHETIX

Customize Your HP-28, *by W.A.C. Mier-Jedrzejowicz*

Written for all users of HP-28 calculators, whether a 28C or the new 28S. This book shows you tips to use your HP-28 more effectively, and tailor it to your exact needs. The powerful but obscure SYSEVAL command is explored in detail, as it is the "trap door" through which you will gain full control of your HP-28. Machine language programming, internal hardware layout, and instructions on expanding the memory of the HP-28C are all included. *228 pages, softcover*

SKWIDBC -- The HP-41 Barcode Generation Module, *by Ken Emery*

If you have access to either a Laserjet^{tm} or a Thinkjet^{tm} printer, you can now instantly and economically produce HP-41 barcode in any size you want. All types of HP-41 barcode are supported. You cann swap programs by mail with anyone that has an optical wand, or use barcode as the ultimate backup memory system! *ROM and complete instruction manual, $199.95*

SKWIDBC Plus *by Ken Emery*

A super speed version of SKWIDBC, for users of the HP LaserJet Plus or Series II printers. (SKWIDBC+ creates a soft font, so it does not work with the original Laserjet.) Also compatible with the Thinkjet. *ROM and instruction manual $249.95, or upgrade your SKWIDBC for $50.00 plus tradein*

Advanced Programming Tips for the HP-41, *by A. McCornack & K. Jarett*

Serious programmers will appreciate this book on efficient programming. Modular programming is explained, along with ways to save program bytes and running time. Several important synthetic programming techniques not covered in other books are discussed, including TEXT 0 prefix key assignments, Catalog 1 crash recovery techniques, and making programs PRIVATE without peripherals. A special feature is line-by-line analyses of all the synthetic programs in **HP-41 Synthetic Programming Made Easy** and **HP-41 Extended Functions Made Easy**. HP-41 Machine code (M-code) is described, so you can decide whether this exciting area is for you. Several application routines for the ZENROM and CCD modules are explained in detail. *340 pages, bar code for programs, softcover, $20.95*

Control the World with HP-IL, *by Gary Friedman*

Use that old HP-41 that's sitting idle for an entirely new kind of project. Team it with an HP-IL to build and control such diverse items as an intelligent telephone autodialer/answering machine (complete with a speech synthesizer!), an automated photographic darkroom controller, an ultrasonic distance measurement unit, a slide projector dissolve controller for two projectors, and more. Imagine being able to accomplish all this with your battery powered, portable, calculator/computer.

The photos, illustrations, and circuit diagrams throughout the book are easy to follow. The general-purpose building blocks are so clever that you'll find dozens of your own uses for them. *340 pages, bar code for programs, plastic spiral binding, $24.95*

Extend Your HP-41, *by W.A.C. Mier-Jedrzejowicz*

Over 650 pages of information, this is the ultimate reference book for your HP-41 system. This book leads you from the basics of keyboard operation and programming, to the latest advanced techniques. A few of the many topics are efficient use of flags and looping instructions, integration to infinite limits, and random number generation. The Advantage module, the Time Module, Extended Functions, and Extended Memory and other modules are described with useful hints. An appendix lists all the known "bugs" of the HP-41 system, and ways for you to determine which ones your system has.

Extend Your HP-41 is by far the largest and most complete collection of useful facts on the HP-41 system. Beginner or expert, no HP-41 owner should be without it. *Over 650 pages, bar code for programs, concealed plastic spiral binding, $29.95*

HP-41 MCODE for Beginners, *by Ken Emery*

MCODE is the internal machine code used by the HP-41, and programs in MCODE run 7 to 120 times faster than programs in the normal user code. This book makes the previously mysterious world of MCODE understandable to any user who enjoys programming an HP-41 calculator. Learn how to build a Function Address Table, and continue from there in easy steps. Also included are application programs and detailed appendices for MCODE experts. NOTE: Additional equipment for your HP-41 is required to do MCODE programming. *190 pages, plastic spiral binding, $24.95*

Price reduced for Closeout sale!

HP-71 BASIC Made Easy, *by Joseph Horn*

Fascinating applications to help you calculate confidently using CALC mode and command stack. General tips on keyboard BASIC, and details on how to use PEEK and POKE are included. A 20 page syntax guide lists hundreds of HP-71 keywords for quick reference. **HP-71 BASIC Made Easy** is an excellent tutorial and an essential reference. *164 pages, plastic spiral binding, $9.95*

Inside the HP-41, *by Jean-Daniel Dodin*

Unlock the secrets of your HP-41! This book is a wonderful sampler of many techniques for the HP-41. The concepts of both synthetic programming and M-Code programs are introduced. Learn the geography of the status registers, and how to manuipulate them to your advantage. *264 pages, softcover, $12.95*

HP-41 Extended Functions Made Easy, *by Keith Jarett*

This book helps you maximize from your extended functions and extended memory. All 14 extra HP-41CX functions are included, most of which can be simulated on the HP-41C or CV. Also presented are powerful application programs: a mailing list manager, text editor, HP-16 simulator, solve, integrate, and utility programs. *Over 250 pages, bar code for programs, plastic spiral binding, $16.95*

HP-41 Synthetic Programming Made Easy, *by Keith Jarett*

Synthetic programming is the creation and use of instructions that cannot be keyed up by normal means. Applications of synthetic instructions include 21 additional display characters expanding USER mode key assignment capability and using the ALPHA register as three data registers. Besides its utility, synthetic programming is just plain FUN. Includes a Quick Reference Card, a $2.00 value.
192 pages, bar code for programs, plastic spiral binding, $16.95

HP-41 Synthetic Quick Reference Guide (SQRG), &
Quick Reference Card for Synthetic Programming (QRC)

Both are musts for synthetic programming, "indestructible" and sized to fit neatly in an HP-41 case. *SQRG $5.95, QRC $2.00*

ORDER BLANK

	Price per copy	Qty	Amount
For HP-32S **Tips and Programs for the HP-32S** by W.A.C. Mier-Jedrzejowicz	$9.95	____	_____
For HP-28C's & HP-28S's **Customize Your HP-28**, by W.A.C. Mier-Jedrzejowicz	$16.95	____	_____
Closeout! Price Reduced! For HP-71'S **HP-71 Basic Made Easy**, by Joseph Horn	$9.95	____	_____
For HP-71'S & HP-41'S **Control the World with HP-IL**, by Gary Friedman	$24.95	____	_____
For HP-41'S **HP-41 Advanced Programming Tips**, by A. McCornack & K. Jarett	$20.95	____	_____
HP-41 M-Code for Beginners, by Ken Emery	$24.95	____	_____
Inside the HP-41, by Jean-Daniel Dodin	$12.95	____	_____
Extend Your HP-41, by W.A.C. Mier-Jedrzejowicz	$29.95	____	_____
HP-41 Extended Functions Made Easy, by Keith Jarett	$16.95	____	_____
HP-41 Synthetic Programming Made Easy, by Keith Jarett (Includes one Quick Reference Card)	$16.95	____	_____
Quick Reference Card for Synthetic Programming	$2.00	____	_____
Synthetic Quick Reference Guide (SQRG)	$5.95	____	_____
ROM's **SKWIDBC -- Barcode Generation Module** by Ken Emery	$199.95	____	_____
SKWIDBC Plus -- for LaserJet Plus or Series II (Upgrade from SKWIDBC for $50 plus SKWIDBC tradein)	$199.95	____	_____
AECROM by Redshift Software	$ 99.00	____	_____
Sales tax (California orders only, 6 or 7%)			_____

Shipping	1st book	Add'l books
within USA, book rate (4th class)	$1.50	$0.50
USA 48 states, United Parcel Service	$2.50	$1.00
USA, Canada, air mail	$3.00	$1.50
elsewhere, book rate (6 to 8 week wait)	$2.00	$1.00
elsewhere, air mail	$12.05 for **Extend Your HP-41**, $6.05 for others	

Free shipping for **ROM's**, **QRC** plastic cards or **SQRG** (any number)

Enter shipping total here $_____

Total due $_____

Checks must be in U.S. funds, and payable through a U.S. bank.

Name _____

Address _____

_____

City _____ State _____ Zipcode _____

Country _____

Mail to:
SYNTHETIX, P.O.Box 1080, Berkeley, CA 94701-1080, USA Phone (415) 428-9009

CONVERSION FACTORS

1 radian	1/2 pi circle
	0.1592 revolutions
	57.3 degrees
	3,438 minutes
	2.06E5 seconds
1 ft	0.3048 m
	3.05E5 μ
1 Å	10E-8 cm
	1E-4 μ
1 Btu	2.93E-4 kw hr
	3.93E-4 hp hr
	0.252 kg cal
	1055 joules
	1.055E10 ergs
	778 ft/lb_f
1 centipoise	0.01 g/cm sec
	6.72E-4 lb/ft sec
	2.42 lb/ft hr
1 lb	4.464E-4 long ton
	5E-4 short ton
	4.54E-4 metric ton
	0.454 kg
	454 grams
1 psia	0.068 atm
	2.31 ft H_2O
	6.895 kPa
	51.72 mm Hg
	0.07703 kg/cm^2
1 lb/ft^3	0.016 g/cm^3
1 ft^3/min	0.125 gal/sec
	0.4720 liter/sec
1 ft/sec	0.682 mile/hr
	1.097 km/hr
	18.29 m/min
	30.48 cm/sec

Useful Constants

c, speed of light in a vacuum	2.99792458E8 m/sec 186,000 miles/sec
N, Avogadro number	6.02252E23 molecules/gmol
G, gravitational constant	6.670E-11 Nm^2/kg^2
e, electron charge	1.60210E-19 coulomb
m_e, electron mass	9.1091E-31 kg
h, Planck constant	1.05459E-34 J/sec 6.624E-27 erg/sec
R, gas constant	8.3141 J/gmol K 0.0821 atm liter/gmol K 1.987 gcal/gmol K 1.987 Btu/lbmol R 10.73 (lb-force/in^2) ft^3/lbmol R
Velocity of sound	(in dry air, 0 deg C, 1 atm) 33,136 cm/sec 1089 ft/sec
Heat of fusion of water	(at 1 atm, 0 deg C) 79.7 cal/gram 144 Btu/lb
Heat of vaporization of water	(at 1 atm, 100 deg C) 540 cal/gram 972 Btu/lb

TIPS AND PROGRAMS FOR THE HP-32S

A Handbook for Science and Engineering

The HP-32S is a remarkable new calculator - and this book will help you make the most of it! There are four chapters leading you step by step from quick tips and routines to detailed programs to solve scientific and engineering problems. Professionals and students alike will appreciate the explanations of programming techniques and the programs themselves.

Chapter 1 presents a series of tips on efficient use of the HP-32S, both from the keyboard and in short program sequences. The detailed comparison of the HP-32S to other calculators makes this book worth buying as an investment guide to selecting the best calculator for your needs.

Chapter 2 has short programs to solve typical problems from science and engineering plus explanations of the programming techniques used. Chapter 3 presents longer programs and more advanced techniques. The powerful SOLVE/INTEGRATE function of the HP-32S is discussed to allow you to make full use of this extraordinary feature. Chapter 4 lists sources of further information.

An added value is the table of commonly used engineering constants and conversion factors

ISBN 0-937637-05-X

50995

9 780937 637050

$9.95