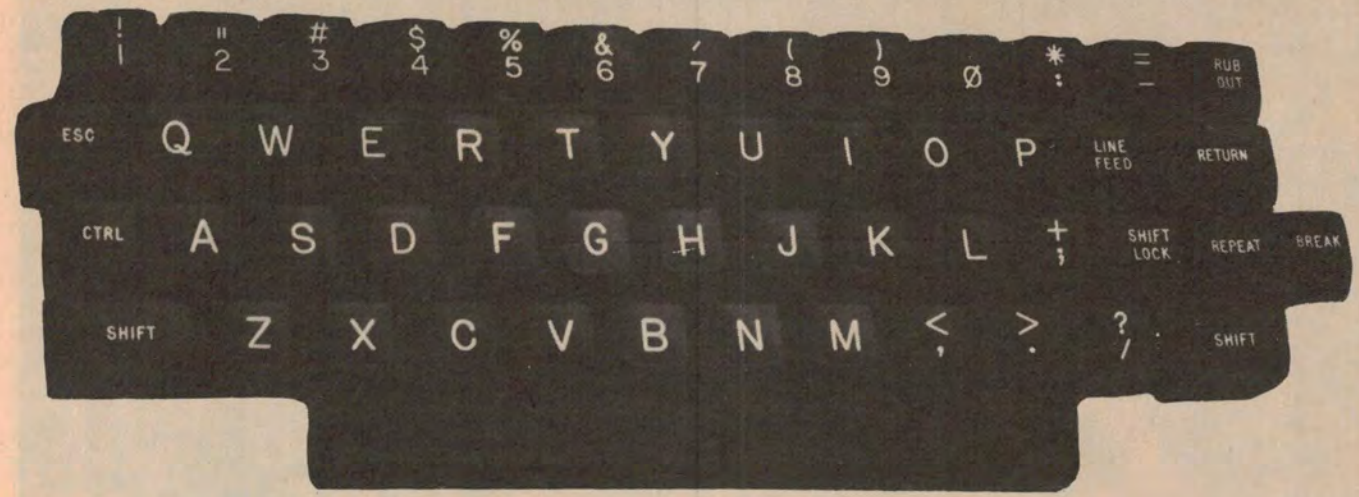


Back door into BASIC Part 2.

Phil Cohen

In the first part of this series Phil Cohen explained the similarities (and some of the differences) between a computer and a calculator. This month he goes on to show how a computer can be used as a calculator — using BASIC.

THE FIRST BASIC word which has to be learned is 'PRINT'. This causes the machine to output information onto the VDU screen or display. The information which the computer will output is determined by what the user puts in to the right of the word PRINT.

Say you type in 'PRINT 4'. The computer would show what you typed as you typed it, then it would reply (on the line below) with the answer, '4'.

At this point, it is worth mentioning the 'RETURN' key. This is a key at the extreme right-hand end of the keyboard, which is the same place as the *carriage return* on an electric typewriter. When you press the RETURN key it is a way of telling the computer to "read what has just been typed in".

Although the computer will show what is being typed in by putting it on the screen, it will not act on it until the RETURN key is pressed.

So, if you type in 'PRINT 4', you then press RETURN. The computer will look at 'PRINT 4', see that the word PRINT means to output whatever is to the right of it, decide to output the value 4, and reply with '4'.

By the way, it may have struck you that, logically, the word should have been DISPLAY, rather than PRINT. The reason for using PRINT is that, at the time that BASIC was developed, teletypes (digitally controlled typewriters) were much more often used than TV-type displays. When the computer PRINTed something in those days, it really *printed* it!

What happens if we put in something a bit more complex? What about 'PRINT 3+4'. The computer will reply with '7'. The computer has looked at what was to the right of the word PRINT, found it to be more than just a simple arithmetic value, worked out the answer and printed that.

Let's look at some more of the BASIC arithmetic functions. If we type 'PRINT 7−6', the answer is '1'. No surprises there.

What about division? 'PRINT 8/2' will give '4'. Notice that the ÷ (divide) symbol is not used. The '/' symbol means that same, and is more commonly found on typewriters and computer keyboards.

Similarly, the '×' symbol is not used in BASIC for multiplication. It has been dropped from computer languages for the same reason it is dropped in algebra — it is easily confused with a lower case x, which is a commonly-used variable name. The symbol for multiplication in BASIC is '*'. So'PRINT 3*4' will give the answer '12'.

Priority of evaluation

Now we come to something which is not usually a problem in calculators. Say we type in 'PRINT 3+4*2'. On most calculators, this will give the answer 14. Not so in BASIC. In BASIC, as in most computer languages, the answer will be '11'.

The reason is that the computer has done the multiplication first. 4 times 2 is 8, and 8 plus 3 is 11. This is in keeping ▶

Personal computing for professionals — the HP-85 computer from Hewlett Packard, designed for personal use in business and industry by professionals such as engineers, scientists, accountants and investment analysts, features powerful central processor, typewriter-like keyboard with 20-key numeric pad, high resolution CRT display, thermal printer, cartridge tape drive, enhanced BASIC language, and interactive graphics in a fully integrated system the size of a portable electric typewriter.

with a general rule which is used in almost all computer languages. Once learned by the user, it makes computer arithmetic much easier to use than calculator arithmetic.

The rule goes like this: Work things out in the following order —

- First see if there are any brackets and work out the bits inside the 'deepest' set of brackets first, then the next deepest, and so on.
- Work out any trigonometric and other complex functions next, such as sine, cosine, log, etc.
- Work out any arithmetic which calls for powers or roots.
- Do any multiplication or division.
- Lastly, do the addition and subtraction.

Within these rules, the computer will work from left to right. so if there are a string of additions and subtractions it will do the left-most one first. If there are brackets in a ( ) ( ) structure, it will work out the left-hand set first.

Some examples will make things a bit clearer.

'PRINT 3*4−2*3

will cause the computer to work out 3 times 4, then 2 times 3, then 12 (the result of the first part) minus 6 (the result of the second part), giving an aswer of '6'.

The BASIC symbol for exponentiation (or raising to a power) is an 'up-arrow' — ↑ . This is used in the following way: If you want to find 2 to the power 3, then, rather than writing 2^3, as you would in mathematics, you instead write 2 ↑ 3. (If you like, the up-arrow shows that the next number is to be shifted up one space).

'PRINT 6.7 + 3 ↑ 2' would cause the computer to work out 3 to the power 2, then add the answer to 6.7, giving the result '15.7'. Notice that the exponentiation was done *before* the addition, in accordance with the rules.

Other Functions

What about functions such as sine and log? In BASIC, these more complicated functions are written as a word to the left of a number in brackets. For example, 'PRINT SIN(17)' will give the sine of 17 radians. Notice that in most cases, BASIC trigonometric functions work in radians and BASIC logs are natural logs. No space is allowed between the N and the start of the brackets, by the way.

In the above example, we used the BASIC word for sine. The BASIC word for cosine is COS and for logarithm is LOG. Not all that difficult to remember. EXP gives the power of e — natural antilogs, if you like.

The number inside the brackets need not be a simple arithmetic value. COS(5-2) will give the cosine of 3 radians.

BASIC Functions

There are many other functions in BASIC which use a name in front of a set of brackets. Some of the more common arithmetic ones are given in the table here. There are various other types which we will deal with in due course.

Variables

I said previously that a computer has memories in which it can store numbers

BASIC FUNCTIONS

Function	BASIC Name	Description
Absolute	ABS	Gives a positive value if the number is negative. For example, ABS(8) is 8, but ABS (−5) is 5.
Arctan	ATN	Gives the inverse tangent. This function is often found even where the computer is not provided with a tangent function, as the tangent of an angle can be worked out easily from the sine and cosine.
Cosine	COS	Gives the cosine.
Exponential	EXP	Gives e to a particular power.
Integer	INT	Gives the integer part of a number. For example, INT(8.97) would be 8.
Log	LOG	Gives the natural logarithm (base e).
Sign	SGN	Gives a value which shows the sign of the number: SGN(−8) is −1, SGN(−4) is −1, SGN (9) is 1, and so on. SGN(0) is 0.
Sine	SIN	Gives the sine.
Square root	SQR	Gives the square root.
Tangent	TAN	Gives the tangent.

in much the same way as a calculator can. In computer jargon, these are called 'variables' because any given part of the memory can store any value — and the value may change ('vary') during the course of the program.

In a computer, the memory is not committed to storing numbers. Parts of it can also be used for storing the program — the memory in a computer is completely 'general purpose'. For this reason, when the machine is turned on, none of the memory contains variables.

If the computer is then told to store a particular value, it will first allocate a small part of its memory for the storage of that value. These small 'pigeon holes' in the computer memory are allocated as required, and each is given a unique name.

In BASIC, the names given to the various memory 'allotments' (or 'variables') take the form of a letter of the alphabet followed by a digit. For example, inputting 'A1 = 3' into the computer will cause it to allocate an area of memory for the storage of one variable, call that area of memory "A1", and then store the value 3 in it.

In BASIC the "=" sign means 'replaced by' rather than the familiar 'equals'. Thus, 'A1 = 3' means 'replace the value in A1 by 3'. A subsequent input of, say, 'A1 = 9' will cause the computer to put the value 9 into A1. This will replace the original value of 3, by the way.

Figure 1 shows what happens during a typical memory transaction. At the start, none of the memory is allocated to any particular variable name. This is in fact a bit of a simplification, as part of the memory is used for program storage and other tasks which we will go into later.

The first input of 'A1 = 4' causes the computer to look around its memory for variable A1, and finding that it does not exist yet, to allocate it a space in memory. It then fills that space with the value 4.

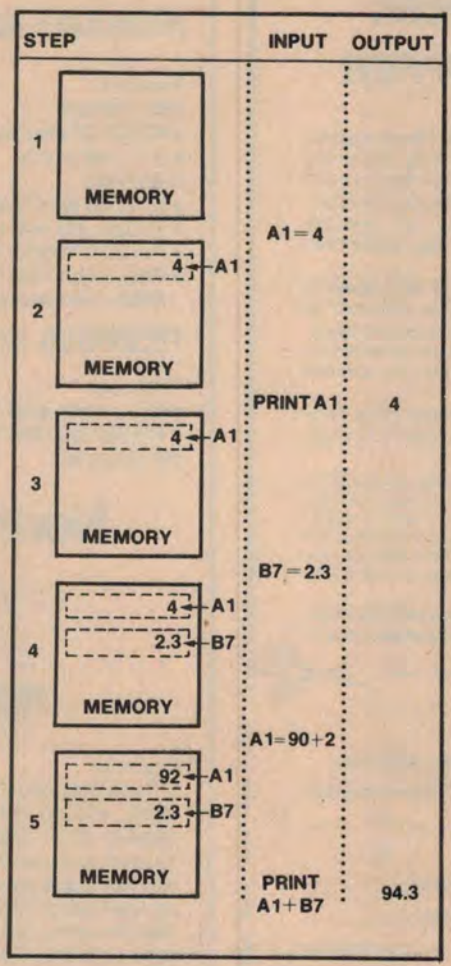


Figure 1. **Memory at various steps during a calculation.**

The next step is the input of 'PRINT A1'. This will cause the computer to look for variable A1 in its memory. It then takes the value that it finds in A1 and outputs it.

If A1 had not existed at this stage, by the way (for example, if you had typed 'PRINT A2' by mistake) then the computer would have been confused. In circumstances like this, most computers output a rather terse message, like 'A1 DOES NOT EXIST'.

The next step is 'B7 = 2.3'. The computer will search for B7 and, finding that it does not exist, will allocate space to it and put in the value 2.3.

The input 'A1 = 90+2' will first cause the computer to work out what 90 plus 2 is. It will then search its memory for A1 and, finding that it *does* exist, will put the value 92 into it (thus obliterating the previous value of 4).

The final step will make the computer search for both A1 and B7 and finding that they both exist, to add their values together and output the result.

By the way, in most versions of BASIC, variable names such as A and B are allowed, so that there are a total of 286 possible names: A, A0, A1, . . . A8, A9, B, B0, B1, . . . Z9.

Memory Usage

Each computer has only a certain amount of memory — it can hold only so many numbers or so much program at any one time. For this reason, when it is necessary for a particular program to use a lot of variables, and where the variables are not going to need fractions, 'integer' variables are used.

Integer means that the number held in that variable can only be a whole number: 2, 45 and −986 are OK, but 5.6 is not.

In BASIC, the way to get an integer variable is to put a % sign after the variable's name. For example, A3% is a valid integer variable. 'A3% = 9' will cause the machine to allocate an area of memory for variable A3%, then put the value 9 into it.

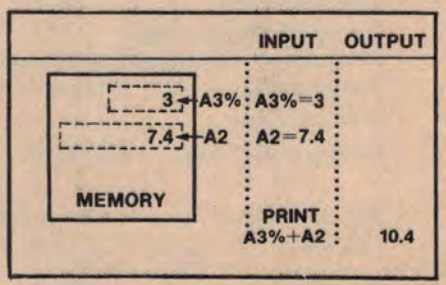


Figure 2. **Integer variables take up less memory.**

Figure 2 shows the use of integer variables. Notice that they take up less room than other number variables.

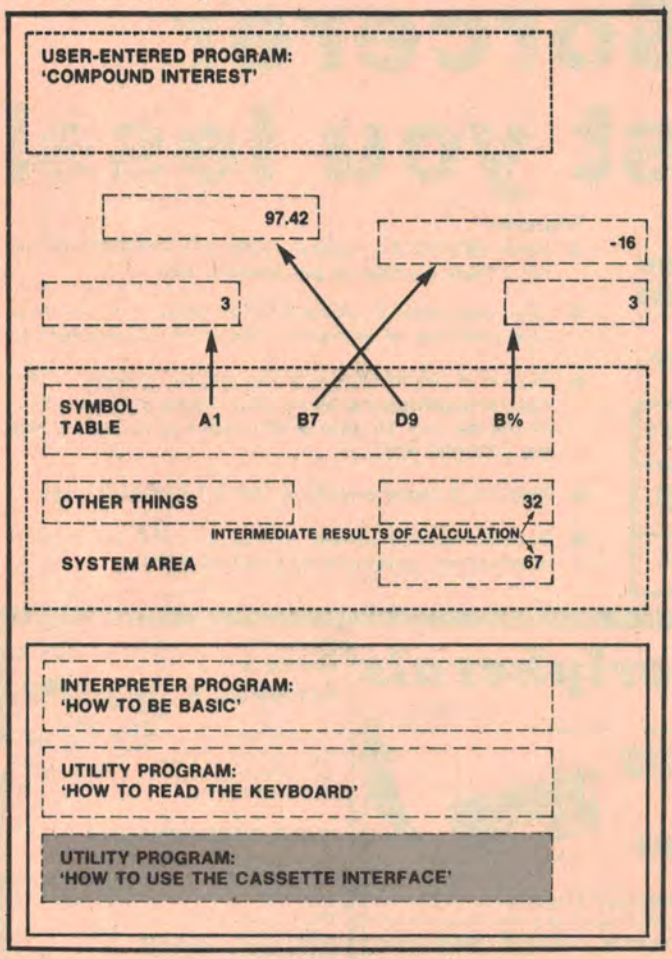
In mathematics, numbers which are not integers are called 'real' numbers. This nomenclature is also used in computing. So we have two types of variables (there are others, which we'll come to later): integer and real. ▶

INSIDE THE MEMORY

The diagram here shows what the inside of a computer's memory looks like to the computer. Although it looks a bit complicated at first, a few minutes contemplation will convince you that it really *is* complicated. That's why I've put it into this box — it's not *absolutely* necessary to know the memory's inner workings in order to be able to program.

It *is* necessary, however, if you want to be able to program well.

The first thing which should spring to notice is the shaded box at the bottom. This is the 'Read-Only Memory' (ROM), and is the only part of the computer's memory which is not cleared when the machine is turned off. This is achieved by using memory chips which have had their contents 'blown' into them. (If you like: engraved on their memories in letters of fire!). This is very useful in that they will remember their contents when the machine is turned on, but with the disadvantage that they then cannot remember anything else.



Inside the ROM are a number of programs not written in BASIC, but in the computer's own language — 'machine code' — which is very difficult for humans to learn, but can be understood by the microprocessor without any help!

There are various 'utility programs' (utility is a word used by the Americans to describe water supply, garbage disposal, swimming pool cleaning and other essentials), which tell the microprocessor how to use the various hardware units connected to it. They tell it, for example, how often to check the keyboard to see if a key has been pressed. They also tell it what format to use when it records programs on cassette, and what format the programs are in when the cassettes are played back.

Also in this ROM is the BASIC 'interpreter'. This is a program (again written in 'machine code') which tells the microprocessor how to read BASIC.

In some machines (such as the Sorcerer) the ROM part of the memory is interchangeable — the ROM chips can be replaced with other ROM chips which tell the microprocessor how to read APL, for example, rather than BASIC.

The rest of the memory outside the ROM can be written into and read by the microprocessor at will. For this reason, it is called 'Random Access Memory' (or by some 'Read And write Memory') — RAM.

At least part of the RAM has to be used by the computer just for its 'housekeeping' tasks — like remembering when it last checked to see if any of the keys on the keyboard had been pressed.

The machine will also store the intermediate results of calculations in system RAM. For example, if it is working out 6*2-5*4, the intermediate result of 6*2 will have to be stored while it works out what 5*4 is.

The 'symbol table' is also stored in the system part of the RAM. This is a list of variables which tell the computer where the values of the variables are to be found. "Why not just label them?", I hear you ask. Well, take the example of the computer looking for the value of B7. If the place where the value of B7 (in this case, -16) was labelled with 'B7', the computer would have to search the whole memory before it was sure of finding it. Using a symbol table, it only has to search the table, then go straight to where the value is kept.

Note that the integer variable takes up less room.

In many computers, the system area used by the machine has a movable boundary — the less it needs, the less it will take up. This means that more is available for other uses.

The rest of the memory is available for any use that is required of it. In the above diagram, four variables are shown, but it is just as possible to have 4000 — if the memory is big enough.

At the top of the diagram is a BASIC program. This calls for variables A1, B7, D9 and B%. Every time the program is run, the computer will clear its symbol table (which *effectively* clears the numbers stored in memory), then as it comes to the first time A1 is mentioned in the program, it will enter it into its symbol table, set aside a place in memory for it and put in the value the program calls for.

The 'other things' mentioned in the system RAM will be dealt with in due course.

Some versions of BASIC (particularly those for very small machines) *only* use integer variables. In this case, it's a means of reducing the complexity of the 'interpreter' (the program which is permanently stored in the machine which tells it how to read BASIC). Integer-only BASICs usually *don't* use the % method of signifying integers — *all* of the variables are going to be integers anyway, so there's no possibility of confusion.

Figures 1 and 2 are a little incomplete; they don't show the 'system' part of the memory. This includes the interpreter (in fixed memory — or 'Read-Only Memory' (ROM)) and any temporary storage the machine may need in order to operate. For example, there must somewhere be a 'symbol table' for the BASIC. This is a table (in the sense of a table on a printed page) which holds the names of all the variables, and where they are to be found in memory.

If you haven't understood *all* of this part of the series — don't worry. Probably the best way to approach it is to put the article down for a day or two, let the ideas settle in, then re-read it.

It's only necessary to understand the bare bones of the internal workings of a computer to be able to program one. It is necessary to understand it in some depth if you want to be able to program well, however. ●

• *Next month, Phil Cohen looks at string handling — which forms the basis of the word processor.*