Can you talk the same language as your home computer?

Understanding BASIC

Never tried to program a computer before? Don't worry, just about all of the new personal computers are programmed in Basic, a language that's surprisingly easy to understand and use. Here's an easy to read primer on Basic and how you use it.

by PETER A. STARK

Basic is probably the most popular and widely used computer language for small computer hobbyists, and with good reason — it is powerful, yet simple. I'm sure you're ready to learn more about what Basic is and what it can do.

Originally invented at Dartmouth College in the USA early in the 1960s, it was intended to bring the computer to the average Dartmouth student in a way that had never been tried before. The traditional approach was to place a large computer centre in the corner of the campus and then force students to go to the centre to run their computer programs.

Dartmouth tried the exact opposite. It placed computer terminals throughout the campus, even in dormitories, within easy reach of every student. It then tempted students to use them, not only by having the terminals easy to get to but also by having a simple computer language to program the computer with. That was the beginning of Basic!

Unlike earlier languages such as Fortran or Cobol, which were intended for large programs, Basic was intended for the small user. A Fortran and Cobol user had to prepare his programs on punched cards away from the computer. Only when he had the entire set of cards ready, would he go into the computer room and enter the cards into the computer. His problem would be run on the computer, his results printed or punched back into cards, and then he would be encouraged to leave to make room for the next user. In other words, these languages kept the user away from the computer as much as possible.

Basic, on the other hand, was designed for use with terminals, such as teletypewriters, which were connected to the computer and actually using the computer for extended periods of time. A student could sit down at a terminal and play a game against the computer for hours on end.

To make this entire idea practical in the days of million-dollar computers required the use of *time-sharing*, where dozens or perhaps even hundreds of terminals were connected to the computer at the same time. Since the computer is so fast, it easily could take care of many students using the computer at the same time, with each student having the impression he was the only user.

But now, the entire approach has changed. For just a few hundred dollars, you can buy a small computer which can run Basic programs. Since it is no longer necessary to time-share, the computer system can be quite simple and cheap, and yet still be powerful enough to run sizable programs, although only one at a time.

To see just what Basic is and what we can do with it, let's sit down at a computer terminal and type in some commands, seeing what the computer does. These examples were run on a Southwest Technical products MP-6800 home computer, but would be much the same with any small home or "personal" computer system.

The first thing we notice is that the terminal has a keyboard similar to a typewriter, except some symbols are in new places and some keys have symbols not found on a typewriter.

For instance, above the comma is the symbol <, and above the period is the symbol >. Of special importance is a key labelled CR or RETURN, which means carriage return. This key means you are finished with a line and want to return the carriage, the part that prints on the paper, to the left, ready for the next line. Every line you enter into the computer must be followed by a CR to tell the computer you are finished it.

Some computers have a key called ENTER, which works very much like the CR key. The Tandy TRS-80 uses an ENTER key, for example.

Let's sit down at the terminal and start by pressing CR (or ENTER). The computer responds with

READY

Like this Apple-II machine, most personal computers are programmed in easy to learn BASIC language. (Courtesy Computerland).

A typical personal computer keyboard. As you can see, it is very similar to that of a typewriter. (Courtesy Dick Smith Electronics).

In our case we get the message

```
READY
#
```

which tells us the computer is ready, and waiting for a command. Let's enter a simple program telling the computer to print something:

```
10 PRINT 2+3
```

A one-line program like this is about as simple as you can get. Don't forget the CR or ENTER at the end. This simple program tells the computer to add 2 and 3 and print the result.

Notice the number 10 in front. Every instruction of a Basic program must have a *line number* before it, and this is line number 10. The 10 does not necessarily mean that this is the tenth line of a program; it just means that we have decided to give this line the number 10. We could have just as well numbered it 1 or 500. The point behind line numbers is that every line of a program has a different line number, so at some later time we can go back and remove or change lines at will, referring to them by line numbers.

Once we have typed in a program such as this one, we can do two things with it — get a listing of it on the printer to check that we have typed it correctly, or run it. To get a listing, we type the word

```
LIST
```

and, as soon as we hit the CR, the computer responds with

```
0010 PRINT 2+3
READY
#
```

With minor changes, the computer simply types the program as we have entered it.

Notice that, up until now, we have not seen the actual answer of 5, which the computer is supposed to print. We merely have entered the program and checked it. To actually perform it and get our answer, we type

```
RUN
```

and the computer prints

```
5
READY
#
```

Notice that there is a difference between *lines of a program*, which *always* get a line number, and *commands* to the computer telling it what to do with the program, which *never* get a line number. The commands we use most often are LIST and RUN, but each computer system has a number of other commands such as:

- NEW or CLEAR — Erase the program
- SAVE — Save the program on tape or other storage for later use
- LOAD — Load a program previously saved back into the computer

Let's erase the simple program we wrote and enter a new one:

```
NEW
READY
# 10 LET I = 3
# 20 LET J=I + 17
# 30 PRINT I, J
#
```

With one exception, every program instruction starts with a short word such as LET or PRINT right after the line number. The one exception is that with many computers, the word LET may be omitted. Notice that each line has a line number. We could have numbered the lines 1, 2, 3 but this is a bad habit to get into. Very often we find, after trying to run the program, we made a mistake and have to add a few lines. With lines numbered 10, 20, 30, and so on, it's easy to slip in extra lines such as line 15 or 18. Even though we may enter them at a later time, giving them a line number between 10 and 20 will automatically tell the computer that we want them placed in that order.

In the above program, lines 10 and 20 mean just what they say. Line 10 says to let a number I be equal to 3. We have to learn the difference between *constants*, which are constant and never change, and *variables* which can vary and change. In this line, the number 3 is a constant while I is a variable. We could, for example, insert another line into the program as follows:

```
22 LET I=5
```

I thus changes — it was equal to 3 at line 10, but becomes equal to 5 at line 22. We could now get a listing of the program as follows:

```
LIST
0010 LET I=3
0020 LET J=I+17
0022 LET I=5
0030 PRINT I, J
READY
#
```

Notice how the computer automatically put line 22 in the right place, between 20 and 30.

Constants are plain numbers such as 3, 5, 17, or -12.597. There is a way of expressing very large or very small constants by using powers of 10, but that need not concern us at this point. By their very nature, they obviously never change.

Variables, on the other hand, are represented by letters such as I or J. In fact, any of the letters A through Z can be used for variables. Since this only would allow 26 different variables, Basic also allows variables to be represented by a letter followed by a number from 0 through 9. This is very convenient for calculations on electrical circuits, since the values of resistors can be represented by the variables R1, R2, and so on.

Let's take the above program and run it.

```
RUN
5     20
READY
#
```

To understand what has happened, we have to examine the above program line by line. Line 10 told the computer

to let the variable I equal 3. Line 20 says to add I (which is 3) to 17, and let J be the answer. Thus J becomes equal to 20. Then, line 22 says to let I equal 5. From this point on, I is 5, not 3, so that line 30 prints 5 for I and 20 for J.

As you can see, the computer performs these instructions in the order of their line numbers, not in the order we typed them in. This is another important use of line numbers — they specify the order in which the computer will perform its instructions.

The opposite of a PRINT statement is an INPUT. For an INPUT, the computer stops, prints a ? prompting character, and then waits for you to type in something. Let's write a short program to allow you to type in a number, have the computer multiply it by 3, and print out the answer. First erase the old program:

```
NEW
READY
#
```

Now enter a new program:

```
#10 INPUT N
#20 S = 3* N
#30 PRINT S
```

Line 10 allows you to type in a number, which becomes the variable N. Line 20 multiplies it by 3 to produce another variable, S; notice how a star * is used to mean *times*. Finally, line 30 prints out the product S. If we now type:

```
# RUN
```

the computer prompts with

```
?
```

and if we supply a number, such as

```
1.2
```

the computer comes back with

```
3.6
READY
#
```

This would not be much fun if we could only enter and print numbers, but Basic also allows us to use letters and words. For example, let's add the line:

```
# 5 PRINT "TYPE IN A NUMBER AND I
                WILL MULTIPLY IT BY 3"
```

and change line 30 to read

```
# 30 PRINT "THE ANSWER IS", S
```

If we list it, we get the printout

```
# LIST
0005 PRINT "TYPE IN A NUMBER AND I
WILL MULTIPLY IT BY 3"
0010 INPUT N
0020 S = 3* N
0030 PRINT "THE ANSWER IS", S
```

Now try running it:

```
# RUN
TYPE IN A NUMBER AND I WILL
MULTIPLY IT BY 3
? 7
THE ANSWER IS 21
READY
#
```

As you can see, enclosing a message in quotes (") and placing it in the PRINT statement makes the computer print it exactly as it stands.

Another type of variable is the *string variable*. It is generally signified by a letter A through Z, followed by the $ sign. Its function is to hold a string of letters or other characters from the keyboard, but allow them to be changed, like variables, throughout a program. To illustrate, let's try a new program:

```
# NEW
READY
# 10 PRINT "WHAT IS YOUR NAME?"
# 20 INPUT N$
# 30 PRINT N$, "IS A NICE NAME"
```

Line 20 lets us input a string of letters, while line 30 prints them out again. watch what happens when we run the program:

```
# RUN
WHAT IS YOUR NAME?
? PETE
PETE      IS A NICE NAME
READY
```

After inputing the name PETE, the computer printed it out again, followed by the words IS A NICE NAME. There is a large space after PETE which is put in by the computer because Basic normally prints its output spread out across the page to be in nice columns if numbers are being printed. in this case it makes the output look messy, but that is easy to get around if we use a semicolon (;) in line 30 instead of a comma. This is one of the fine points in Basic, which are of little interest to the beginner but are very useful to the expert.

Next month we will take this sample program a little further, and use it to show how you can get your computer to do things a controlled number of times. We'll also have a look at the ways it can be told to make decisions.

Programming your computer in BASIC is easier than you thought!

(Continued next month)

Reprinted from the March 1979 issue of "CQ" magazine, by kind permission of the author and Cowan Publishing Corporation.