How to talk the same language as your computer — 2

Understanding BASIC

Here is part 2 of the author's easy to read introduction to the Basic programming language, used on just about all of the new personal computers. He shows you how to get your computer to do things a controlled number of times, and how to get it to make decisions.

by PETER A. STARK

The tremendous power of the computer comes from the fact that programs, or portions of them, can be repeated over and over. Suppose we add one more line to the last program we tried in part 1:

```
#10 PRINT "WHAT IS YOUR NAME?"
#20 INPUT N$
#30 PRINT N$, "IS A NICE NAME"
#40 GO TO 30
```

and run it again:

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

and so on . . .

Computer experts would now say the computer is *stuck in a loop*. It would keep on printing out the same line over and over if we didn't stop it by pushing a button on the control panel. Our last line, line 40, is the culprit. it told the computer to go back to line 30 and repeat from there. Thus the computer does the printout in line 30, and the very next line sends it right back to do another printout, and so on. This is an *infinite loop*, since it never stops — unless we push a button to stop it, that is.

A better way of controlling a GO TO is with an IF instruction. For example, we can say IF X=3 GO TO 30, and the GO TO will only be done by the computer if the value of the variable X happens to be 3.

Let's change the above program so it will ask for a name, and will only print out "IS A NICE NAME" if the name happens to be PETE; otherwise, the computer will answer that the name is a poor one:

```
#10 PRINT "WHAT IS YOUR NAME?"
#20 INPUT N$
#30 IF N$ = "PETE" GO TO 60
#40 PRINT N$, "IS A POOR NAME"
#50 GO TO 10
#60 PRINT N$, "IS A NICE NAME"
#70 GO TO 10
```

As before, the computer asks WHAT IS YOUR NAME. If you answer PETE, then line 30 tells the computer to go to line 60, so that it will print the name again, followed by the words IS A NICE NAME. For any other name, the computer will *not* go to line 60, but will instead continue to line 40 and print IS A POOR NAME. Either way, a GO TO 10 returns to the top, so the computer asks for another name. Let's run it to see what happens:

```
#RUN
WHAT IS YOUR NAME?
? SAM
SAM       IS A POOR NAME
WHAT IS YOUR NAME?
? GEORGE
GEORGE    IS A POOR NAME
WHAT IS YOUR NAME?
? PETE
PETE      IS A NICE NAME
WHAT IS YOUR NAME?
?
```

As before, the computer is stuck in a loop since it keeps returning to step 10. This is usually not quite what we want.

A good loop is one which has an end to it. In some way, we like to tell the computer when to get out of the loop. One common way is to count the repetitions of the loop, and stop at some predetermined number of them. For example, the following program prints out the numbers from 1 to 12 and their squares:

```
#NEW
READY
#10 LET N = 1
#20 LET S = N*N
#30 PRINT N, S
#40 LET N = N + 1
#50 IF N < 13 GO TO 20
```

Line 10 starts the number N at 1; line 20 squares it by multiplying it by itself; line 30 then prints the number N and its square S. Now, line 40 says something a bit different from what a mathematician would expect from N = N + 1 (which is not really a good equation after all.) What it means is that the computer should take the value of N, add 1 to it, and then place the result back as a new N. In other words, line 40 adds 1 to N. Since N started at 1, it is now 2. But since this is in a loop, in a little while N will go to 3, and then 4, and so on, all the way up to 12.

The symbol < in line 50 means *less than*, so this line says "if N is less than 13, go back to line 20." But eventually N will go from 12 to 13, and when that

BASIC is also used to program many small microcomputer development systems, like this one from the Italian firm SGS-Ates.

Many of the small computer systems now starting to be used in business are programmed in BASIC, like the one shown here. (Courtesy Computerland)

happens, line 50 no longer sends the computer back to line 20. So we have here a loop which is repeated exactly 12 times.

The IF statement is very useful, since it allows checking whether two things are equal or not. In addition to the less than or < symbol, we also use > which means *greater than*. The combination <>, means *less than or greater than*, which is the same as saying *not equal*, so IF X <> 5 GO TO 300 means that if X is not equal to 5 the computer should go to line 300. Moreover, instead of ending the IF with a GO TO, we can also end with the word THEN followed by any other valid Basic instruction. Our program to judge whether a name is nice or not could have been written with these two IFs:

```
#40 IF N$ = "PETE" THEN PRINT N$,
"IS A NICE NAME"
#50 IF N$ <> "PETE" THEN PRINT N$,
"IS A POOR NAME"
```

Two other combinations are <= which means *less than or equal*, and >= which means *greater than or equal*.

The idea of using a variable to count the repetitions of a loop is so common and useful that Basic has a special pair of instructions just for that purpose — the FOR and NEXT pair. These always go together, the FOR at the start of the loop and the NEXT at the end. To see how they work, let's rewrite the program to square the numbers from 1 to 12:

```
#NEW
READY
#10 FOR N = 1 TO 12
#20 LET S = N * N
#30 PRINT N, S
#40 NEXT N
```

Line 10 tells the computer that N is the counter, and it is supposed to vary from 1 to 12. Initially, N starts at 1, and the computer continues down through the following steps until it gets to NEXT N. Now it adds 1 to N, and goes back to the first statement inside the loop, which is line 20. It will repeat the loop, adding 1 to N each time, until N reaches 12. When N tries to go to 13, the loop ends.

There is a variation on the FOR which lets N change in different ways; this is done by adding one more word to the line:

```
#10 FOR N = 1 TO 12 STEP 1
```

This specifies that N is supposed to go from 1 to 12 in steps of 1. If we said

```
#10 FOR N = 1 TO 12 STEP 3
```

then N would go up in steps of 3. Or if we said

```
#10 FOR N = 12 TO 1 STEP -1
```

it would go from 12 back to 1 in steps of -1. That is, N would go 12, 11, 10, 9, 8, and so on, all the way to 1. Just to see what happens, let's try running the program:

```
RUN
12   144
11   121
10   100
9    81
8    64
7    49
6    36
5    25
4    16
3    9
2    4
1    1
READY
#
```

Basic has several more possible instruction types. Some, like REM (remark) and STOP, are useful to the beginner and we will see them later in some of the demonstration programs. Others are for more advanced users and we will skip them here.

In addition to the various instruction types, Basic also has *functions* which perform specific maths calculations or some other operations. For example, a mathematician or engineer might use the SIN or COS functions when working with angles. The functions likely to be used by the beginner, out of the dozen or more most computers have, are these:

- INT ( ) converts whatever is placed inside the parenthesis into the next lower integer (whole number). For example, saying

```
#10 LET J = INT(3.14)
```

would make J equal to 3.

- RND (0) makes the computer invent a random number between 0 and 1. This is usually used in games, for coming up with random moves or random numbers. For instance,

```
#10 LET J = RND(0)
```

would result in J becoming equal to some unknown value between 0 and 1.

Sometimes we combine the RND and INT functions to generate other random numbers. For instance, suppose we are writing a game where the computer is supposed to pick a card from a deck of cards and print out what it is. Since there are 13 cards in a suit, we need a random number which is a whole number between 1 and 13.

If we use RND to make a number from 0 to 1, and then multiply it by 13, the result will be a number from 0 to 13. Add 1 to this, and you have a random number between 1 and 14, but always just a bit smaller than 14. Convert it to an integer with INT, and you have a whole number ranging from 1 to 13 (and never equal to 14.) The result of putting all this into one line is

```
#100 LET C=INT (RND(0) * 13+1)
```

One more function useful to beginners is the TAB ( ); which makes the terminal's printer or display move over to the right to the position indicated by whatever is inside the parenthesis. For example

```
#50 PRINT TAB(15); I
```

would print the value of I fifteen places from the left end of a line on the printer. Note that the TAB is used in a PRINT statement, and that it is usually followed by a semicolon.

Finally we are ready to put all this together into several simple programs. How about a program to pick five cards at random and print out what they are? We will program it as a loop which is repeated five times, use the RND func-

tion to pick a random number, and use IF statements to print out words like JACK or KING:

```
#NEW
READY
#10 FOR I=1 TO 5
#20 LET C=INT(RND(0)*13+1)
#30 IF C<11 THEN PRINT C
#40 IF C=11 THEN PRINT "JACK"
#50 IF C=12 THEN PRINT "QUEEN"
#60 IF C=13 THEN PRINT "KING"
#70 NEXT I
```

Let's see how this runs:

```
#RUN
1
7
QUEEN
7
2
READY
#
```

Now let's add a few more steps to add the suit. We will use RND again to pick a number between 1 and 4, and use it to print out the suit. Add the following steps:

```
#25 LET S=INT(RND(0)*4+1)
#62 IF S=1 THEN PRINT TAB(6); "OF HEARTS"
#63 IF S=2 THEN PRINT TAB(6); "OF DIAMONDS."
#64 IF S=3 THEN PRINT TAB(6); "OF CLUBS"
#65 IF S=4 THEN PRINT TAB(6); "OF SPADES"
```

To see what the program now is, we list it:

```
#LIST
0010 FOR 1=1 TO 5
0020 LET C=INT(RND(0)*13+1)
0025 LET S=INT (RND(0)*4+1)
0030 IF C<11 THEN PRINT C
0040 IF C=11 THEN PRINT "JACK"
0050 IF C=12 THEN PRINT "QUEEN"
0060 IF C=13 THEN PRINT "KING"
0062 IF S=1 THEN PRINT TAB(6); "OF HEARTS"
0063 IF S=2 THEN PRINT TAB(6); "OF DIAMONDS"
0064 IF S=3 THEN PRINT TAB(6); "OF CLUBS"
0065 IF S=4 THEN PRINT TAB(6); "OF SPADES"
0070 NEXT 1

READY
```

OK, let's run this:

```
#RUN
KING
        OF HEARTS
5
        OF DIAMONDS
JACK
        OF CLUBS
6
        OF DIAMONDS
JACK
        OF CLUBS
READY
#
```

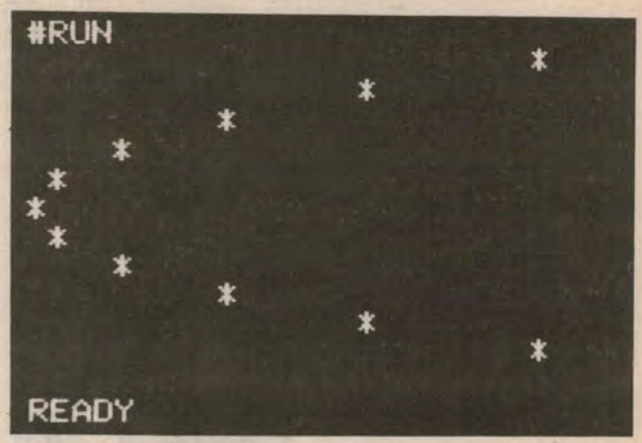

Fig. 1: By using the PRINT TAB command, you can get your computer to plot a simple graph, like this parabola.

We could neaten the output so each card is printed on one line, but that's more complicated. Let's do another example. How about a program to input the names of two people and print them out in alphabetical order?

```
#NEW
READY
#10 PRINT "ENTER TWO NAMES"
#20 INPUT A$, B$
#30 IF A$<B$ THEN PRINT A$, B$
#40 IF B$<A$ THEN PRINT B$, A$
#RUN
ENTER TWO NAMES
? SMITH, JONES
JONES    SMITH
READY
#
```

Notice how we are comparing two strings of letters as if they were two numbers; whichever is less is printed first. Although this example only sorts two names, we could do it for more names with a more complicated program.

Suppose a math student needs to plot an equation for his homework. The equation is $y=x^2-10x+26$, and he is supposed to find y for x going from 0 to 10. This program would do it:

```
#NEW
READY
#5 REM THIS IS A REMARK
#7 REM LET X GO FROM 0 to 10
#10 FOR X=0 TO 10
#20 LET Y=X*X -10*X+26
#25 REM PRINT BOTH X AND Y
#30 PRINT X, Y
#35 REM END OF LOOP
#40 NEXT X
#50 REM WHEN LOOP IS DONE, STOP
#60 STOP
```

Note the remark (REM) lines. These are purely for our own convenience — the computer ignores them when it actually runs the program. Now let's see the program run:

```
RUN
0        26
1        17
2        10
3         5
4         2
5         1
6         2
7         5
8        10
9        17
10       26
READY
#
```

Better yet, why not have the computer plot a graph? Change line 30 to

```
30 PRINT TAB(Y); "*"
```

Now when we run the program, we get the result shown in Fig. 1. Not bad for a simple little program, is it?

The graph may be sideways and a little coarse, but it certainly gives the picture.

Well, I hope this introduction has given you an idea of how easy it is to program a computer in Basic. It's really not hard at all when you get the hang of it. Now why don't you try writing a few programs of your own?

Don't worry if a program doesn't work first time you try it. Just keep trying — before long, you'll be making your computer jump through hoops with the best of them!

SUGGESTED FURTHER READING

If you want to go a bit further into Basic language programming, there are two books which can be thoroughly recommended. Both are written by Dr David Lien, of San Diego:

BASIC COMPUTER LANGUAGE (TRS-80 Level 1 User Manual), published by Tandy Corporation. Available from all Tandy stores for $5.95

THE BASIC HANDBOOK, published by Compusoft Publishing, San Diego. Available from Dick Smith Electronics stores, also the Technical Book and Magazine Company. Price $15.95.

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