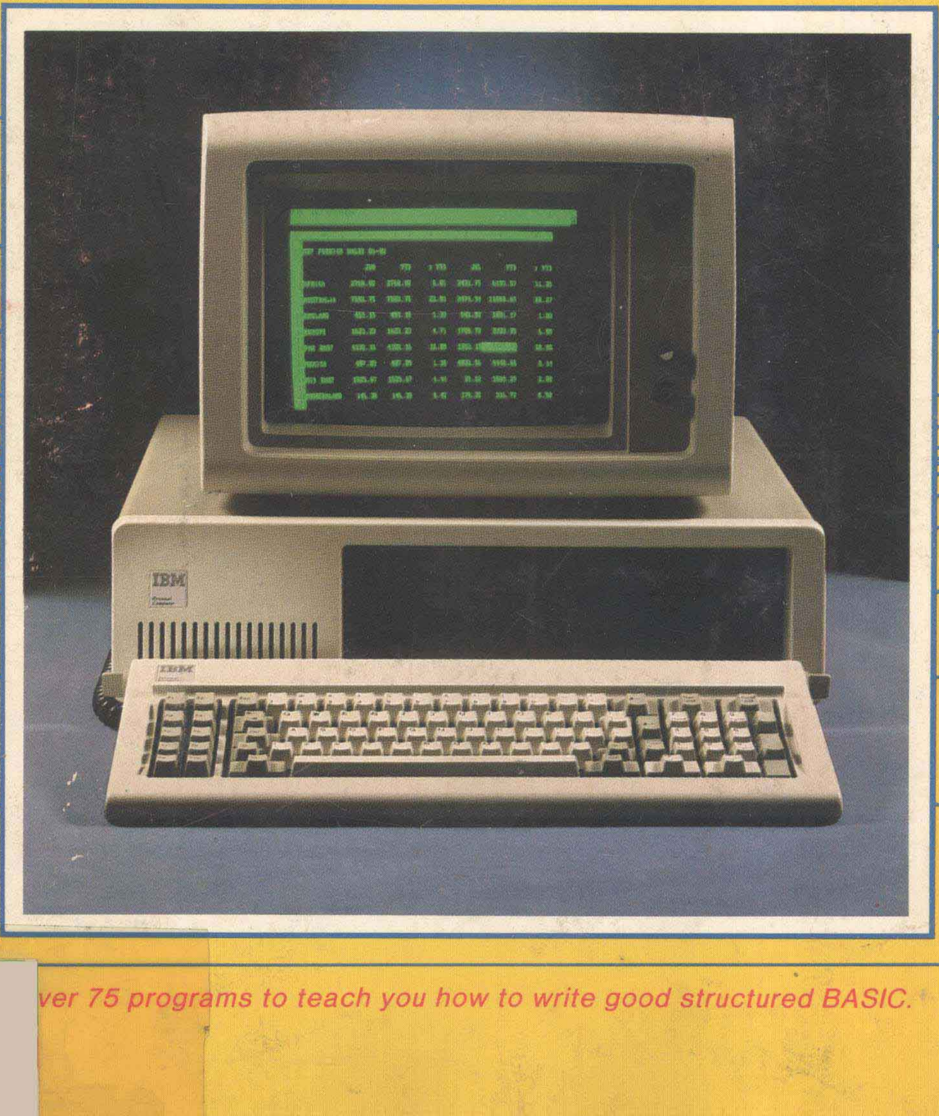
MICROPOWER SERIES

TECHNIQUES OF BASIC for the IBM Personal Computer

John P. Grillo / J. D. Robertson



ver 75 programs to teach you how to write good structured BASIC.

MICROPOWER SERIES

TECHNIQUES OF BASIC for the IBM Personal Computer

John P. Grillo / J. D. Robertson
Bentley College
Waltham, Massachusetts

wcb
Wm. C. Brown Publishers
Dubuque, Iowa

Consulting Editor:
Edouard J. Desautels
University of Wisconsin-Madison

Cover photo by Bob Coyle

Library of Congress Catalog Card Number: 83-71136

ISBN 0—697-08276—8
2—08276—01

Printed in the United States of America

Third Printing, 1983

2-08276-03

To Paul and Evans

Introduction

BASIC has come a long way since its first days at Dartmouth College in 1964, when because of its simplicity it helped students to learn about the computer. It has evolved in two stages. The first stage occurred in the early 1970s when minicomputers became standard fixtures in many small business, scientific, and educational environments. At that time BASIC became more than a curiosity. Because of its expanded features, particularly file management, it began to appear as the application language of choice for the popular minis.

The second stage of BASIC's evolution is occurring right now. The popularization of the microcomputer in the early 1980s has resulted in BASIC being the *de facto* standard as a high-level language for these new devices. Remember that minis were used primarily in small businesses, scientific labs, and schools. The micros have come into the home, and BASIC has come with them. Suddenly the phrase "computer power to the people" means something tangible to millions of individuals. The decade of the 1980s will end with a substantial fraction of the public actively involved in developing programs for their personal use, and most of these programs will be written in BASIC.

All this is fine, as long as this tool is used for its intended purpose, that being to entertain, to educate, to calculate, and to manage files. However, many purchasers of microcomputers will bring it home, play a few games of Blackjack, Chess, or Star Trek, and perhaps maintain a recipe file. This is not enough. These devices are more powerful than the million-dollar computers of the 1960s, and to use them for such trivial tasks is only to waste their true potential. It's as if you were to buy a TV set and leave it tuned to a single channel. Microcomputer power should be explored and exploited to its fullest, and one way you can do so is to use it for more than the repetitive execution of one or a few programs. Program it yourself.

As educators we have exposed many students to the joys of computer programming, and we are continually surprised at the variety of people who exhibit a talent for this science, or art, or craft. No general rule seems to apply; programming talent seems to appear in a large and unpredictable segment of the population. The microcomputer revolution will add greatly to the growing numbers who know how to write programs. A few of these people will become excellent programmers. Our aim with this book is to increase the ranks of better programmers by exposing them to some techniques for solving problems that are commonly found in a wide variety of applications.

When you write a program, remember that you must consider three different points of view.

1. The *programmer* is the originator of the program, its creator. In many situations, you will find no existing program that even

remotely begins to solve your problem. This is when your skill as a programmer is tested to its fullest. You are most of all a problem solver at this stage, and your major task is to decide on the method of solution, or algorithm, for your problem.

2. The *reader* of your program is very possibly also its author, but may also be someone else who wishes to adapt it to his or her own application. A program's reader must understand the fundamentals of the language about as well as the programmer, but is rarely involved in its original creation. The remarks in a program are intended for its reader. During your program's development, you are also its reader, and you can use the remarks effectively to remind you of the program's logic or to help modularize it for easier alteration.

3. The *user* of your program is the intended target for its application. Usually, that person is naive about computers and programs. The program, its advanced techniques, and its wealth of remarks are lost to the user. But you, the programmer, must always keep the user in mind. Here's one of the few general rules that has no exceptions: All well-written programs are easy to run.

The programs, program segments, and examples contained within this book have been tested on an IBM Personal Computer system with two minifloppy disk drives. The listings were produced on a Brother HR-1 letter-quality printer. We have purposely oriented this book toward the IBM Personal Computer for the following reasons:

1. The IBM Personal Computer (hereafter referred to as the IBM PC), is one of the many microcomputers that use Microsoft BASIC, which is fast becoming a standard of comparison for both performance and variety of extensions.

2. As of this writing, the IBM PC is beyond its early reputation as the new kid on the block. It is the machine that many people have been waiting for. It has become a major force in the industry, with only the Apples and the TRS-80s as serious competition.

3. The IBM PC can have a color display monitor, so its BASIC has a variety of commands to manage that aspect of output. However, many owners have elected to purchase IBM's monochrome display unit. For this reason and also to keep this book to a manageable size, we do not discuss the color, joystick, sound, and light pen instructions that IBM's version of Microsoft BASIC provides.

4. The IBM PC has a wide range of available peripheral equipment. In addition, its popularity has prompted many peripheral manufacturers other than IBM to produce competing hardware, including hard disks with capacities on the order of 20 million characters.

This book is not intended for the rank beginner. It starts right off with the IF—THEN—ELSE two-way branch construct, and fairly flies through its somewhat advanced topics. The assumption we have made is that you already know what we refer to as primitive BASIC; that is, you have made yourself familiar enough with the language that you can write simple programs. You should know about counters, accumulators, loops, simple string and numeric variables, expression evaluation, and simple algorithms. If you feel uncomfortable at this point about your level of background knowledge in BASIC, you should strongly consider purchasing the book *User's Guide with Applications for the IBM Personal Computer*, a companion book in this series offered by Wm. C. Brown.

What this book does, which few if any other books do, is to show you the power and flexibility of this computer language by example. We have written many programs between these two covers, programs that demonstrate the bells and whistles of Microsoft BASIC. Some of these programs are trivial giveaways. Some are large and powerful, the kind for which you would be expected to pay dearly through a software house. We feel that the wisest companion investment with this book is the diskette that contains its programs. Somehow, one doesn't seem complete without the other.

We hope you will try out all of the features that are discussed in this book. Your reward will be a deep understanding of both a fine computer programming language and some excellent programming techniques.

Contents

1

Decisions and Branching

The IF—THEN and the GOTO are certainly simple to learn and understand, but as a person improves in programming techniques, the limitations of these statements become a real burden. This is where the extensions to the language are particularly rewarding. They are very easy to learn and use, and they make any program easier to read.

IF—THEN—ELSE

Primitive BASIC is limited to having only a line number following the THEN, for example:

```
300 IF X=A THEN 820
```

This restriction leads to awkward programs full of GOTOs that force the reader to jump around from one line of code to another. This process of bypassing some lines and tracing the program in various sequences tends to frustrate both the programmer and the reader. As an example of this poor, and all too common type of programming, look at the program below.

```
10 'filename: "LARGEST1"
20 ' purpose: to find largest of 3 numbers (poorly structured)
30 '  author: jdr & jpg 12/82 (car)
40 '
50 DATA 1,2,3,1,3,2,22,11,33,22,33,11,35,15,25,35,25,15,0,0,0
60 'read three values from data block
70 READ A,B,C
80 IF A*B*C=0 THEN 10000
90 'check to see if A is largest
100 IF B > A THEN 140
110 IF C > A THEN 180
120 L=A
130 GOTO 200
140 'check to see if B is largest
150 IF C > B THEN 180
160 L=B
170 GOTO 200
180 'here we know that C is largest
190 L=C
200 'print the value of L; it is the largest
210 PRINT L; "IS THE LARGEST OF"; A; B; C
220 GOTO 70
10000 END
```

```
 3 IS THE LARGEST OF 1  2  3
 3 IS THE LARGEST OF 1  3  2
 33 IS THE LARGEST OF 22  11  33
 33 IS THE LARGEST OF 22  33  11
 35 IS THE LARGEST OF 35  15  25
 35 IS THE LARGEST OF 35  25  15
```

Note: The PRINT command is used here and throughout the book to display output on the screen. To have the output appear on the printer, you should change the PRINT commands to LPRINT.

The program LARGEST1 is difficult to compose, to trace, and to debug. Extended BASIC lets the programmer instruct the computer to do something after finding out that the condition is true, instead of just branching somewhere else. Look at this rewrite of the same program.

```
10 'filename: "LARGEST2"
20 ' purpose: to find largest of 3 numbers (better)
30 '  author:  jdr & jpg 12/82 (car)
40 '
50 DATA 1,2,3,1,3,2,22,11,33,22,33,11,35,15,25,35,25,15,0,0,0
60 'read three values from data block
70 READ A,B,C
80   IF A*B*C=0 THEN 10000
90 ' store the largest in L, then print L
100      IF A>B THEN IF A>C THEN L=A
110      IF B>A THEN IF B>C THEN L=B
120      IF C>A THEN IF C>B THEN L=C
130   PRINT L; "IS THE LARGEST OF"; A; B; C
140 GOTO 70
10000 END
```

Notice that the decisions in this program have no branches. Each IF statement checks the truth of a pair of conditions, and the value of L is set when both conditions within the same statement are true.

But extended BASIC has even more. The THEN clause may be followed by another clause, called the ELSE clause. The resulting compound statement allows the programmer to specify one statement to be executed if the condition is true, and another statement if the condition is false. For example, study the two program segments below.

```
50 'if discriminant D of quadratic equation is non-negative,
60 'then compute and print the roots; otherwise print the
70 'message, "NO REAL ROOTS"
80 D=B*B-4*A*C '       calculate the discriminant
90 D2=2*A '    calculate deniminator of quadratic equation
100 IF D>=0 THEN PRINT "Roots=";(-B+SQR(D))/D2;(-B-SQR(D)/D2
          ELSE PRINT "No real roots"
```

```
200 'let user stop or proceed, but accept only YES or NO answer
210 PRINT "Do you want to go on (answer YES or NO)";
220 INPUT A$
230 IF A$="NO" THEN STOP
            ELSE IF A$<>"YES" THEN 210
```

Logical Operators

This feature allows more English-looking conditional statements by the use of OR, AND, and NOT operators.

```
10 'if A is less than B and A is less than C,
20 'then print A as the smallest.
30 IF A<B AND A<C THEN PRINT A; "IS SMALLEST"
40 IF A$="YES" OR A$="SURE" OR A$="OK" THEN 500
```

Program LARGEST3 shows how much more readable these logical operators are in a program, as opposed to nested IFs or an abundance of GOTOs.

```
10 'filename: "LARGEST3"
20 ' purpose: to find largest of 3 numbers (best)
30 '  author: jdr & jpg 12/82 (car)
40 '
50 DATA 1,2,3,1,3,2,22,11,33,22,33,11,35,15,25,35,25,15,0,0,0
60 'read three values from data block
70 READ A,B,C
80   IF A*B*C=0 THEN 10000
90 'find and print the largest all in one shot
100 'note that the program occupies more space in memory,
110 '  but its operation is very clear
120   IF A>B AND A>C THEN PRINT A; "IS THE LARGEST OF";A;B;C
130   IF B>A AND B>C THEN PRINT B; "IS THE LARGEST OF";A;B;C
140   IF C>A AND C>B THEN PRINT C; "IS THE LARGEST OF";A;B;C
150 GOTO 70
10000 END
```

```
 3 IS THE LARGEST OF 1  2  3
 3 IS THE LARGEST OF 1  3  2
 33 IS THE LARGEST OF 22  11  33
 33 IS THE LARGEST OF 22  33  11
 35 IS THE LARGEST OF 35  15  25
 35 IS THE LARGEST OF 35  25  15
```

The logical operators AND, OR, and NOT can be used for Boolean logic operations. Study this statement:

```
10 ' set A to TRUE (-1) if both conditions are true,
                          otherwise to FALSE (0)
20 A=(X=Y) AND (J>0)
```

The parentheses around each of the conditions is necessary to isolate the conditions from the assignment of the answer to A.

```
10 'set the flag V to TRUE if A is not less than B
20 V=NOT(A<B)
```

This statement sets V to true (– 1) if the statement is true; that is, the value of A is not less than B. Notice that the statement

```
10 V=(A>=B)
```

does the same thing, and it is perhaps clearer.

There are some applications for using purely logical operators. Remember that in these cases the values in question are stored by the computer as either true (– 1) or false (0). Such applications lead to statements like these:

```
10 'set P to -1 if X is 0 and vice versa
20 P=NOT(X)
30 IF J THEN PRINT "TRUE"
40 IF NOT(A AND B) THEN PRINT "NEITHER IS TRUE"
50 IF (A OR B) THEN PRINT "EITHER ONE OR BOTH IS TRUE"
```

A third possible application of logical operators is in bit manipulation or bit comparison. This could be used in a program to identify the positions of the 1-bits in any variable, in effect representing it in binary form. The program BITS exemplifies the problem, converting the variable X that the user entered to its binary representation. The test value T starts at the value 16384, which is two to the fourteenth power. Each time through the loop in lines 100-150, T is reduced by a factor of two, in effect shifting the single one-bit to the right one position.

```
10 'filename: "BITS"
20 ' purpose: display last 15 bits of integer X
30 '  author: jdr & jpg 12/82 (car)
40 '
50 'set T to be the first (largest) power of 2
60 T=16384
70   PRINT "WHAT INTEGER DO YOU WISH CONVERTED (0=STOP)";
80   INPUT X
90     IF X=0 THEN 10000 ELSE PRINT X, "IN BINARY IS  ";
100  FOR I=1 TO 15
110 '          isolate the single bit from X
120    B=T AND X
130      IF B>0 THEN PRINT "1 "; ELSE PRINT "0 ";
```

```
140    T=T/2
150  NEXT I
160  PRINT
170 GOTO 60
10000 END
```

```
WHAT INTEGER DO YOU WISH CONVERTED (O=STOP)? 1
 1            IN BINARY IS  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
WHAT INTEGER DO YOU WISH CONVERTED (O=STOP)? 15
 15           IN BINARY IS  0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
WHAT INTEGER DO YOU WISH CONVERTED (O=STOP)? 3456
 3456         IN BINARY IS  0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0
WHAT INTEGER DO YOU WISH CONVERTED (O=STOP)? 32767
 32767        IN BINARY IS  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
WHAT INTEGER DO YOU WISH CONVERTED (O=STOP)? 0
```

Some of the effects of binary operations using the logical operators can be quite misleading. You should study the explanations of these operations in the BASIC Manual. We include the examples below more for completeness than for clarification. We suggest that you use these operations only if you feel comfortable with binary representation of values in the computer.

Instruction	Output
`1Ø PRINT Ø AND Ø`	Ø
`2Ø PRINT Ø AND 1`	Ø
`3Ø PRINT 1 AND Ø`	Ø
`4Ø PRINT 1 AND 1`	1
`5Ø PRINT Ø ØR Ø`	Ø
`6Ø PRINT Ø OR 1`	1
`7Ø PRINT 1 OR Ø`	1
`8Ø PRINT 1 OR 1`	1
`9Ø PRINT NOT Ø`	-1
`1ØØ PRINT NOT -1`	Ø

ON—GOTO and ON—GOSUB

These closely related branching statements allow a great deal of flexibility when a program needs to perform a multiple-way branch. They both use a variable after the ON, and a series of line numbers after the GOTO or GOSUB. The integer value of the variable is calculated, and a branch is taken to the first statement if the variable is 1, the second if 2, the third if 3, and so on. On the IBM PC, as on most other computers, if the integer value of the variable does not correspond to the position of a given line number, the statement following the ON—GOTO or ON—GOSUB is executed.

Example:

```
50 ON X GOTO 80, 300, 750, 10, 80, 900
60 'fall through if X<0 or X>6
```

The computer obtains the integer portion of X, which must be between 0 and 255.

If the integer portion of X is	then the computer branches to this line number
< 0	ERROR MESSAGE
0	60 (the next line—no branch)
1	80
2	300
3	750
4	10 (notice that the line numbers do not need to be in order)
5	80 (notice that the same line can be reached with different values of X)
6	900
7-255	60 (the next line—no branch)
>=256	ERROR MESSAGE

You could write the equivalent of line 50 above without use of an ON—GOTO like this:

```
50 IF INT(X)=1 THEN 80
51 IF INT(X)=2 THEN 300
52 IF INT(X)=3 THEN 750
53 IF INT(X)=4 THEN 10
54 IF INT(X)=5 THEN 80
55 IF INT(X)=6 THEN 900
60 ..........
```

If the word GOTO was replaced by GOSUB, the multiway branch would become:

```
50 ON X GOSUB 80, 300, 750, 10, 80, 900
62 'fall through if X<0 or X>6
```

which is equivalent to:

```
50 IF INT(X)=1 THEN GOSUB 80 : GOTO 62
52 IF INT(X)=2 THEN GOSUB 300: GOTO 62
54 IF INT(X)=3 THEN GOSUB 750: GOTO 62
56 IF INT(X)=4 THEN GOSUB 10 : GOTO 62
58 IF INT(X)=5 THEN GOSUB 80 : GOTO 62
60 IF INT(X)=6 THEN GOSUB 900
62 ..............
```