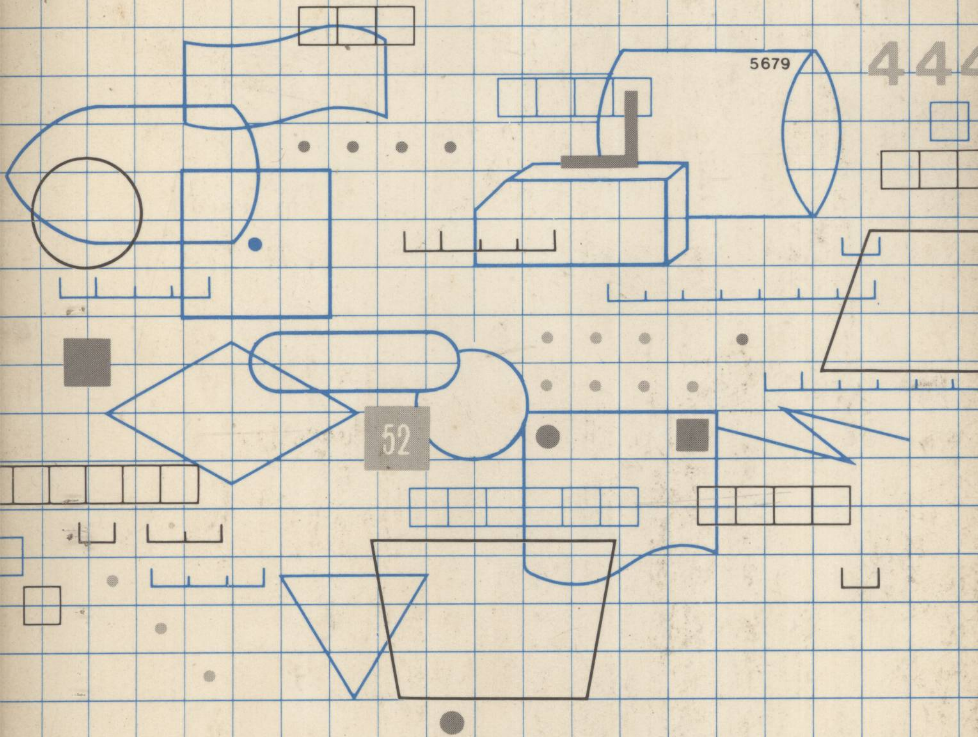
MICROSOFT™ BASIC



Second Edition
By Ken Knecht

TP31
K11
E.2

8461074

Microsoft™ BASIC

Second Edition

Ken Knecht

dP

dilithium Press
Beaverton, Oregon

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Knecht, Kenneth.
  Microsoft BASIC.
  Includes index.
  1. Basic (Computer program language)
I. Title.
QA76.73.B3K57 1982 001.64'24 82-14730
ISBN 0-88056-056-8

Printed in the United States of America

dilithium Press
P.O. Box 606
Beaverton, Oregon 97075

Microsoft™ BASIC

Second Edition

Table of Contents

Introduction

In the early 1970's, the computer hobby really sprang to life with the introduction of the Altair 8800 by MITS. The birth of this first 8-bit hobby computer was prompted by the introduction of Intel's 8080 microprocessor integrated circuit. By the way, when introduced, the 8080 microprocessor was said to cost $350.00. Now it can be purchased for about $6.95.

This early microcomputer ran the BASIC language. Who wrote this early version of BASIC? Why, Microsoft of course. The actual authors were Bill Gates and Paul Allen. With over one million installations, Microsoft Corporation continues to be a major supplier of BASIC interpreters for microcomputers.

This second edition of *Microsoft™ BASIC* describes the latest version of Microsoft BASIC: BASIC-80, release 5.0. While this release of BASIC-80 is the latest version of Microsoft BASIC, some popular microcomputers are supplied with a slightly different version. Fortunately, the differences are not great. If you understand BASIC as described in this book, you will have little trouble adapting most BASIC programs to your computer.

One other popular version of Microsoft BASIC is discussed in this book, that supplied with the Radio Shack TRS-80 Model III. This version is compared with BASIC-80, allows you to see some typical variations from BASIC-80.

To give credit where credit is due, Microsoft did not invent BASIC. This was accomplished by Prof. John G. Kemeny and Prof. Thomas E. Kurtz, who originally designed, implemented and introduced BASIC at Dartmouth College in 1964. BASIC was then implemented on many large computers before hobby computers and Microsoft came along.

There are many other languages available for microcomputers, among them FORTRAN, LISP, C, COBOL, FORTH and Pascal. However, BASIC is the most popular, and most of the programs listed in the many hobby computer magazines are in BASIC.

There are many ways to program computers: in machine language, in assembly language and in high-level languages. Machine language and assembly language (low-level languages) require an intimate knowledge of how your specific computer works. Generally such programs are not transportable. That is, an assembly language program for an Apple will not run on a TRS-80. However, BASIC is a high-level language and it is transportable. A BASIC program written for an Apple will probably run on a TRS-80, with a few slight changes.

A high-level language must be either compiled or interpreted. An interpreter is easier to work with, but a compiled language will usually run the program faster. Most languages are either compiled or interpreted, but not both. An advantage of BASIC is that it can be run either way. That is, frequently a BASIC compiler is offered for a given computer, in addition to the BASIC interpreter. Microsoft offers a compiler for BASIC-80 and TRS-80 BASIC. This compiler is described in Chapter 10 in this book. That chapter also explains in more detail the differences between a compiler and an interpreter. However, the main thrust of this book is on interpreted BASIC, which is what you will most likely be using.

I think you will find programming in BASIC an easy-to-learn, fascinating hobby. The challenges are many, and you will continue to learn more and more about the subtle techniques of advanced BASIC programming as your experience grows. I've been programming in Microsoft BASIC for over eight years and I'm still discovering new tricks of the trade.

Have fun!

Ken Knecht
Yuma, AZ

Chapter One

Definitions

Before we get into programming, it would probably be best to present a glossary of the terms used in this book. This will include some sample commands to illustrate certain points and provide an insight into the way BASIC works. Any commands mentioned in this chapter will be explained in detail in later chapters, so do not despair if the operation or use of a command mentioned is not clearly explained. Also, the bracketed designations of which versions of BASIC support the command will not be used in this chapter.

Terms will be defined in an order which permits later definitions to use terms defined earlier. Therefore, they are not in alphabetical order.

INITIALIZATION: This is the process of loading the BASIC interpreter into memory and defining the parameters in which it will operate.

Details of this process vary considerably depending on the operating system used, so it will be left to you to use the method described in your documentation.

COMMAND: These are instructions to BASIC that are generally, but not necessarily, used in direct mode. By direct mode we mean a command used to operate BASIC directly, without writing a program. When BASIC is ready to accept a command (not running a program) it displays OK (sometimes a > or other prompt is used). This is called command mode. Here's a quick example. After BASIC displays OK, respond with

```
PRINT 4+4
```

This causes BASIC to display 8 and then display the prompt. This is a command. If you entered

```
10 PRINT 4+4
```

nothing would happen immediately. If you then entered RUN on the next line BASIC would then display 8, then the prompt OK again. In this case

```
10 PRINT 4+4
```

is a BASIC program. The line number 10 preceding the word PRINT makes it a program line, and the RUN command must be used to execute the program; BASIC will not print the answer to 4+4 until you enter RUN.

Any instruction to BASIC not preceded by a line number is a command, and will be executed at once. Any instruction preceded by a line number is not a command and will not be acted upon until the program is run. Most instructions can be used either way.

STATEMENT: This is any instruction to BASIC preceded by a line number (see LINE NUMBER). Almost all statements can be used as either commands or statements, depending on whether or not a line number is present.

Normally a statement is an instruction that appears in a program while a command is an instruction used in direct mode (see COMMAND). A statement in a program is said to be in the indirect mode. As mentioned earlier, most instructions can be used either way.

LINE NUMBER: This is a number from 0 to 65529 which appears before each statement in a BASIC program. Some utility programs have problems when you use line number 0, so you might want to avoid using this line number.

Program statements will be executed in ascending numerical order, though they do not need to be entered this way. For example, we write the program

```
10 A=4
20 B=5
30 C=A+B
40 PRINT "FINISHED"
```

This will add 4 and 5, just as you requested, then print FINISHED on the terminal. Unfortunately, only the computer knows how much 4+5 is; we forgot to tell it to display the answer. So we enter

```
35 PRINT C
```

This adds line 35 to the program, between line 30 and line 40, and the answer (9) will now be printed. Since 35 is greater than 30, the computer will calculate the answers in lines 10, 20, and 30. Then it will go to the next line number, 35, even though it was entered after line 40. The program will now display a 9 before it goes to line 40 and displays FINISHED.

So we see the computer will operate on the lines in numerical order, even if we didn't enter them in that order. You can see now why we leave some unused numbers between line numbers, for cases like this. There will be times, not often I hope, when you will wish you had left more than 9 unused line numbers. Later we will see how the renumber command can solve this difficulty.

As we stated earlier, the line numbers can be most any we wish. For example

```
0 A=4
100 B=5
1017 C=A+B
3462 PRINT C
65529 PRINT "FINISHED"
```

This will perform exactly like the preceding program. BASIC just takes the line numbers in ascending order. It doesn't care how many numbers we leave out between them. However, normal programming practice starts the program at line 10 and uses increments of 10 for the following line numbers.

CONSTANT: This is a number used in a program or command. In the previous programs, 4 and 5 were constants. It is a number that the computer accepts as a value in a program; it cannot be changed by the program. The range of acceptable numbers is from 10^{-38} to 10^{+38}. That is, from a number preceded by a decimal point followed by 37 0's to a number followed by 38 0's. For those not familiar with scientific notation, 1×10^5 is the same as 100000 and 2×10^{-8} is the same as .00000002. In BASIC we would enter these numbers as 1E5 or 2E−8.

VARIABLE: This is the symbolic representation of a constant or a number defined by the program. To return to our earlier example

```
10 A=4
20 B=5
30 C=A+B
35 PRINT C
40 PRINT "FINISHED"
```

A, B, and C are variables. In this case, A represents the constant 4, B the constant 5, and C represents the program computed values of A+B, or 9. Variables not representing a constant are given a value of 0 until the program sets their values (see CLEAR). The same variable can represent many numbers in the course of a program run. It always holds the last value set, or 0 if no value has yet been set.

In 8K BASIC a variable name may be any length. However, only the first two characters are significant. The first character must be a letter; the second can be a letter or number. In EXTENDED and DISK BASIC 40 characters are significant. The first character must be a letter; the others can be letters, numbers, or decimal points (periods).

Thus, if the following program were run

```
10 HARRY=1
20 HAROLD=2
30 PRINT HARRY+HAROLD
```

in 8K BASIC the computer would display a 4, in EXTENDED or DISK BASIC the computer would display a 3. In 8K only the HA would be significant, so line 20 resets HA from 1 to 2, and HA+HA=4. In DISK or EXTENDED the two variable names are different, so the result is 3.

Do not use FN as the first two characters in a variable unless you mean it to be a user defined function (see the chapter "ARITHMETIC IN BASIC"). See also RESERVED WORDS in this chapter.

ALPHANUMERIC (character): This is any printable character such as A, L, b, &, + , 9, etc. A letter of course is A to Z, a digit (number) from 0 to 9.

CONTROL CHARACTER: This is a special alphanumeric. It is entered by holding the control key down while a character key is pressed. These are used for some special commands to BASIC and should not be used in variables or program names.

NUMERICS: These are numbers. There are several types of numbers used in BASIC.

The INTEGER is a number from −32768 to 32767 with no decimal point. In 8K BASIC this has no special meaning except for the INT( ) function. In the other BASICs integers can be used to decrease the program's running time and save space. This will be discussed further in other chapters.

In EXTENDED and DISK BASIC an integer variable is designated by adding a % to the end of the variable name. Thus A1% is an integer variable.

SINGLE PRECISION is the type of number normally used in BASIC and is assumed unless a number is specifically declared to be an integer or double precision. Single precision numbers have a precision of 7 digits. Therefore,

```
10 A=1.23456789
20 PRINT A
```

would display 1.23457.

```
10 A=123456789
20 PRINT A
```

would display 1.23457E+08.

```
10 A=999999
20 PRINT A
```

would display 999999, but any greater number would be transformed to the E format. Thus,

```
10 A=1000000
20 PRINT A
```

would display 1E+06. Going to the smaller numbers,

```
10 A=.01
20 PRINT A
```

results in .01 but

```
10 A=.001
20 PRINT A
```

gives 1E–03. See how it works? The computer gives you back your input if the number is between .01 and 999999, otherwise it uses the E format.

In EXTENDED and DISK BASIC an ! can be appended to the end of the variable name to indicate that it is a single precision variable, A1! for example. Normally, however, A1 would be used instead.

DOUBLE PRECISION is found in EXTENDED and DISK BASIC. This format permits 17 digits of precision. The value .01 is still the smallest that can be displayed without going to the E format.

A double precision variable is signified by following the variable name with a #. Thus A1# is a double precision variable.

FLOATING POINT means using the E format mentioned earlier.

STRING: This is a group of alphanumeric characters surrounded by double quotes (" "). Thus, the "FINISHED" of our first example is a string. Strings may be up to 255 characters in length, not including the double quote (" ").

STRING VARIABLE: This is any legal variable name followed by a $. String variables can be used in any version of BASIC. For example,

```
10 A$="FINISHED"
20 PRINT A$
```

would result in FINISHED being displayed. Note that the string must be enclosed in double quotes. Thus, strings cannot include a double quote. Later we will see how we can get around this shortcoming. The surrounding double quotes are not displayed with the string.

See CLEAR for more information about strings.

STRING LITERAL: This is the string enclosed by double quotes. A string can include punctuation, numerals, letters, or anything but a quotation mark. Control characters can be included in a string, but as these are invisible it is best not to use them unless you have to.

EXPRESSION: An expression is two or more variables connected with (an) operator(s). Thus, A + B is an expression.

FUNCTION CALL: This is an intrinsic function of BASIC such as SQR(4), which means find the square root of 4. There are many other functions which will be presented in later chapters.

OPERATOR: This is the +, −, ( ), <>, =, etc., used in a statement or command. Many are used and will be detailed in later chapters.

INTRINSIC FUNCTION: This is the same as a function call, mentioned earlier.

ERROR messages: These are displayed by BASIC when you do something you shouldn't. Dividing by 0, asking for the square root of a negative number, making a syntax error such as X)5 instead of X=5, or many other errors result in error messages. Your BASIC documentation gives a long list of all error messages and their causes. See also Appendix C.

EDIT: This can have two meanings. There is a simple editing which permits you to reenter or change a character while still typing a line (in direct or indirect mode). Another edit feature, only in EXTENDED and DISK BASIC, permits you to return to a program line and replace or insert one or more characters in that line.

RESERVED WORDS: These are used as instructions to BASIC. A reserved word, such as PRINT, cannot be used as a variable in any version of BASIC. In 8K BASIC a reserved word may not be imbedded in a variable name. Thus a variable name such as APRINT or TOTAL would be illegal in 8K BASIC (APRINT contains the reserved word PRINT, TOTAL contains the reserved word TO). A list of reserved words for each version of BASIC is in Appendix B.

SUBROUTINE: This is a part of a program that will be repeated one or more times. It is usually set off from the rest of the program by using higher line numbers than the rest of the program and ends with a RETURN statement. It is used so a sequence of identical statements will not have to be entered