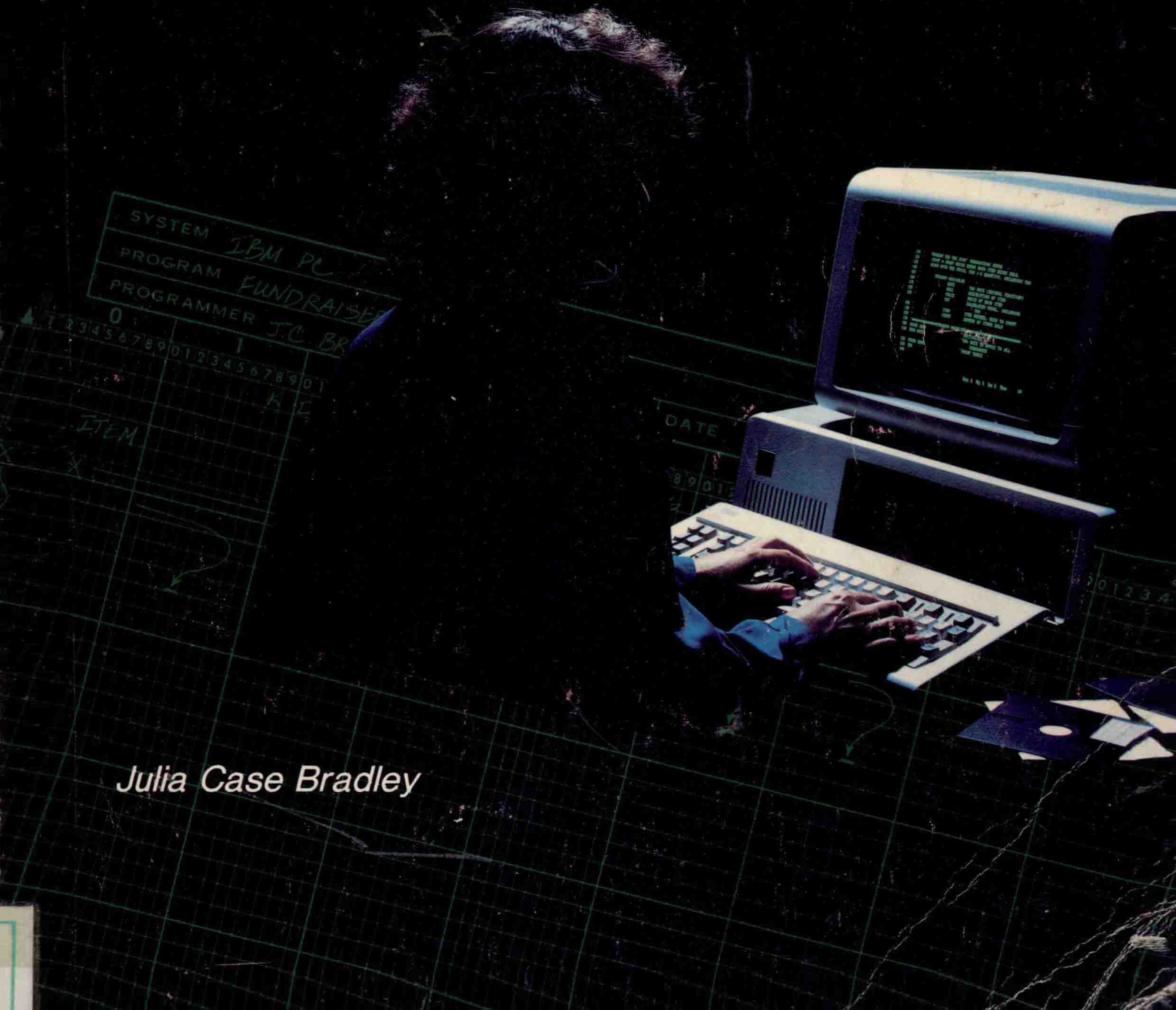
Microsoft® BASIC
Using Modular Structure



Julia Case Bradley

Microsoft® BASIC Using Modular Structure

Julia Case Bradley
Mt. San Antonio College

wcb
Wm. C. Brown Publishers
Dubuque, Iowa

Book Team
Edward G. Jaffe Executive Editor
Nicholas Murray Senior Editor
Nova Maack Associate Developmental Editor
Mark D. Hantelmann Designer
Jean R. Starr Production Editor
Carla Arnold Permissions Editor

Photo Credits **pages 4, 5:** © James L. Shaffer; **page 6:** Courtesy, Apple Computer, Inc.

Cover photograph by Bob Coyle

Library of Congress Catalog Card Number: 85–72275

ISBN 0–697–00455–4

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Preface

This textbook is intended for use in an introductory course in BASIC, which assumes no prior knowledge of computers or programming. The fundamentals of programming are taught in a style consistent with current thinking in the computing field. The student programmer will learn good techniques from the start, rather than having to alter habits that are already formed.

Structured Programming

The primary feature of the text is the development of well-structured, modular programs. Program modules are implemented with subroutines by the early introduction of the GOSUB statement. Appearing throughout the text are complete example programs that are models of good programming style—meaningful variable names; complete remarks including a dictionary of variable names, program mainline and subroutines; clear, consistent indentation; and control structures limited to the three "proper" constructs—sequence, selection, and iteration. The students are *not* taught to program with the GOTO statement at all.

Microsoft BASIC

The dialect of BASIC chosen is Microsoft BASIC. This selection was made for several important reasons. Microsoft BASIC

1. has the statements allowing for the implementation of the three structured constructs. Specifically, with the inclusion of the IF-THEN-ELSE and WHILE/WEND, programs *can* be written without the use of the GOTO.
2. is the most common version of BASIC in use on microcomputers.
3. is the BASIC supplied with the personal computers manufactured by IBM, DEC, Texas Instruments, AT&T, and the many "compatibles," "look-alikes," and "work-alikes."
4. allows formatting of program output with the PRINT USING statement.
5. provides a method for storage and retrieval of data in disk files.

Extensive Appendixes

The inclusion of necessary reference material in the appendixes will do away with the need for most additional, supplementary material generally needed in programming courses. By limiting the dialect of BASIC to that used only on microcomputers running CP/M or MS-DOS, the text can then cover the system commands necessary to operate in those environments. The appendixes include:

1. Edit Mode for Microsoft BASIC (both for the single line editor and the full-screen line editor)
2. BASIC commands
3. CP/M and MS-DOS commands
4. Debugging techniques
5. Discussion and examples of ways to control the special functions of printers
6. Answers to the many feedback questions interspersed throughout the text

7. A list of the reserved words in Microsoft BASIC
8. The ASCII code
9. Error trapping
10. CHAIN and COMMON

Interactive Program Style

The programming emphasis is on interactive program style using menus, good screen design, and input data validation. The INPUT statement is the first method covered for entering data into programs. Not until students are accustomed to interactive programs are the READ and DATA statements covered. Since the majority of software implemented on microcomputers is interactive, it makes sense to learn programming in this manner.

Data File Handling

This text goes well beyond most in the area of data file handling. Chapter 12 covers file concepts and gives extensive coverage of sequential files, including creation, retrieval, and appending data. Chapter 13 covers random data files and shows examples of random and sequential retrieval, updating, and reporting. In chapter 14, indexed files are discussed, and a complete example included illustrating the creation and maintenance of an indexed file. Sequential file updates are also covered in Chapter 14.

Complete Chapter Summaries

At the conclusion of each chapter is a comprehensive list of topics covered in the chapter. During extensive classroom testing of the manuscript, this was one of the most popular features. Although a few of the more advanced students read *only* the summary, most students used the summaries to review the material, for reference to look up topics and terms, and to study for exams. Professors found them a concise source of exam questions.

Program Planning with Flowcharts, Pseudocode, and Hierarchy Charts

The important topic of program planning is covered completely using the three most popular planning tools. Although the trend in the industry is toward dropping the use of flowcharts and switching to either pseudocode or hierarchy charts (or a combination of the two), many students benefit from the visual presentation of program logic afforded by flowcharts. By including all three methods, the student can select the method most beneficial to him or her—and the professor has the choice of methods to use in class.

Significant Changes (improvements) in the traditional sequence of topics

1. Early coverage of subroutines.
   Early (chapter 2), programs are broken into modules using subroutines. Throughout the text programs are written in a modular style, stressing the concept of good module design.
2. Coverage of looping before selection.
   The concept of conditions is presented in the context of controlling loop execution rather than selecting alternate courses of action. The student can easily grasp simple program loops without the necessity of the IF-THEN and IF-THEN-ELSE. With this order of topics, programs can be more meaningful sooner.
3. Usage of the WHILE/WEND for most program loops.
   Until chapter 9, all program loops are formed with the WHILE and WEND. The FOR/NEXT is introduced just before array handling, mainly as an aid to subscript manipulation. Any loop that must be terminated early is always coded with WHILE/WEND, and *never* does execution branch out of a loop with a GOTO or IF-THEN branch.

4. Early coverage of PRINT USING and LPRINT USING.

   The first programs involving program loops (chapter 3) also include formatting the output with (L)PRINT USING. In this way, the student programmer may create pleasing, properly aligned output.
5. Early coverage of structured programming guidelines.

   The first program examples are coded in a consistent, structured style without giving the complete rationale. Then, as soon as the student has learned enough to understand the terms (chapter 5), a complete coverage of structured programming and top down programming is presented. This is far superior either to having a late chapter (after the student has developed a programming style) or an early chapter (before the student can understand the terms) devoted to structured programming concepts.

Numerous Examples and Programming Exercises Each chapter has one complete programming example, showing the program planning, program coding, and output. Additionally, many smaller examples are included throughout the text.

Feedback Questions and Exercises Interspersed at appropriate points are thought-provoking questions and exercises to test student learning. Answers are found in an appendix.

BASIC Statements, Commands, and Functions Boxed As each new statement, command, or function is introduced, it is enclosed in a box with its complete format. It is also explained and illustrated with examples, which clearly demonstrate its use. These boxes serve as a source of easy reference for the student.

Coverage of Output Report Design A rare topic for BASIC programming textbooks is planning reports. The use of printer spacing charts is covered, along with multiple page output.

Flexibility of Use This text is ideally suited for the variety of programming courses currently being taught (and those being considered). That "one semester course in BASIC" is far from standardized, and many colleges and universities are beginning to offer a second semester of BASIC programming. In many cases, BASIC is introduced in a computer concepts course, and in others, the programming is in an independent course.

The coverage of this text is actually more comprehensive than that of most one-semester courses. This gives the professor some latitude in the selection of topics. Here are some possible variations, using parts of the material presented.

1. Short course for students with no background.

   Spend two or three hours on chapter 1 to give the student an understanding of the hardware and software concepts. Leave out the material on data files and sorting. The course would terminate with chapter 11.
2. Concept courses which include BASIC.

   Skip chapter 1 (which covers the fundamentals being more completely covered). Use chapters 2, 3, 4, 5, 8, FOR/NEXT from 9, and 10.
3. One semester (or quarter) course that has a prerequisite.

   Skip chapter 1.
4. File oriented course.

   Cover chapter 13 (random files) early. Some successful courses cover this material before chapter 4, which allows programs to have data file input even before the introduction of selection (IF-THEN).

5. Advanced course.

   The concepts in chapter 7 (interactive programming, screen formatting, data validation) are commonly taught in a second BASIC course. Also, sequential updates and indexing random files (chapter 14) are topics often found in second semester programming courses. Some of the materials in the appendices make ideal lessons for an advanced course. Error trapping and CHAIN and COMMON statements (appendix I) are generally considered advanced topics, as well as controlling printer functions (appendix H).
6. Mathematically oriented course.

   Cover chapter 9 after chapter 2. Chapter 9 includes the FOR-NEXT statements, functions, and the DEF FN statement.

Supplementary Materials This text is part of a complete package of course materials.

1. Instructor's manual. Includes chapter outlines, teaching suggestions, test questions, and exercise solutions.
2. Instructor's diskette. Includes a working solution to many programming exercises.
3. Testpak. Computerized version of the test questions contained in the Instructor's Manual.

Acknowledgments

I would like to express my appreciation to the many people who have contributed to the successful completion of this text. Most especially, I want to thank my colleagues and the students of Mt. San Antonio College who helped class test the material in the text and greatly influenced the final form of the manuscript. I am also grateful to the reviewers who gave constructive criticism and many valuable suggestions.

Bryan J. Carney—University of Wisconsin, Eau Claire
Maurice Eggen—Trinity University
Norman Lindquist—Western Washington University
Lewis D. Miller—Cañada College
Lawrence J. Molloy—Oakland Community College
Donald L. Muench—St. John Fisher College
Pasha A. Rostov—California Polytechnic State University at San Luis Obispo
R. Waldo Roth—Taylor University
Sherman Sherrick—William Jewell College
Charles Williams—Georgia State University

Summary of BASIC Statements

Statement	Effect	Text Page #
MID$(stringvariable, start position, number of characters) = stringexpression	Replaces part of a string variable with the string expression.	169
NAME "old filename" AS "new filename"	Renames a diskette file	321
NEXT [loop index]	Terminates a FOR...NEXT loop.	217
ON ERROR GOTO line number	Enables program error trapping and specifies the line number of the error handling routine.	423
ON numeric expression GOSUB list of line numbers	Evaluates the numeric expression and begins execution of the subroutine specified.	196
ON numeric expression GOTO list of line numbers	Evaluates the numeric expression and branches to the corresponding line number.	436
OPEN mode, #filenum, "filename"	Opens a sequential data file. Mode must be "I" for input files or "O" for output files.	310
OPEN "filename" FOR mode AS #filenum	Alternate format. Opens a sequential data file. Mode must be INPUT, OUTPUT, or APPEND.	310
OPEN "R", #filenum, "filename", reclength	Opens a random data file.	327
OPEN "filename" AS #filenum, LEN = reclength	Alternate format. Opens a random data file.	327
OPTION BASE number	Sets the minimum value for array subscripts. Must be 0 or 1.	250
PRINT items to print	Displays data on the screen. Items to print may be separated by semicolons or commas (with different results).	22
PRINT USING "string literal"; items to print PRINT USING stringvariable; items to print	Displays data on the screen, according to the format specified in the string literal or variable.	76

应用模结构的 Microsoft BASIC

Summary of BASIC Statements

Statement	Effect	Text Page #
GOSUB linenumber	Begins execution of a subroutine. The RETURN statement returns from the subroutine.	45
GOTO linenumber	Branches to the specified line number.	417
HOME	Apple MBASIC. Clears the screen.	160
HTAB position	Apple MBASIC. Moves the cursor to horizontal column position on the line.	162
IF condition THEN statement(s) [ELSE statement(s)]	Performs the statement(s) following THEN when the condition evaluates *true*. Performs the statements following the ELSE (if included) when the condition evaluates *false*.	94
INPUT ["prompt";] variable [,variable2,...]	Inputs data from the keyboard. If a prompt is included, it prints before the input occurs.	41
INPUT #filenum, variable [,variable2,...]	Reads data from a sequential data file.	314
KILL "filename"	Deletes a file from diskette.	321
LET variable = expression	Evaluates the expression and assigns the result to the variable.	33
LOCATE row, column [,cursor]	Places the cursor at the row and column position specified. The optional cursor parameter controls the visibility of the cursor.	160
LPRINT items to print	Prints data on the printer. Items are separated by commas or semicolons.	22
LPRINT USING "stringliteral"; items to print LPRINT USING stringvariable; items to print	Prints data on the printer, according to the format specified in the string literal or variable.	76
LSET stringvariable = stringexpression	Left-justifies the string expression in the string variable.	330

Contents

1 Introduction to Computers and BASIC

RUN Command
LIST and LLIST Commands
SAVE Command
LOAD Command
FILES Command
KILL Command
AUTO Command
RENUM Command
PRINT and LPRINT Statements

Upon completion of this chapter, you should be able to:

1. Describe the computer and its functions.
2. Explain the difference between hardware and software.
3. Differentiate between application software and operating system software.
4. Understand the functions performed by interpreters and compilers.
5. Become familiar with the steps in program design and development.
6. Draw flowcharts using standard flowcharting symbols.
7. Learn to code, enter, and test an elementary BASIC program.

The Computer

The computer, like a typewriter or calculator, is a tool for solving problems. Once mastered, it can be made to perform marvelous feats on command. Without mastery, it can be like a typewriter or calculator in the hands of a child who has not learned its use—its great potential goes to waste.

Many things must operate together for a computer to do any useful work. Assume you have just purchased a powerful, flexible, accurate, obedient robot. You want the robot to work for you—perhaps scrub the floor, fix the car, wash the dishes, or do math homework.

This robot will do anything you ask, but it doesn't know *how* to do anything for itself. It has no common sense. If you tell the robot to change the spark plugs in the car, it won't know that first it must open the hood. If you remember to tell it to open the hood, you had better be sure to place the steps in the correct sequence. Otherwise, it will attempt to change the spark plugs and *then* open the hood. (At this point, hope that the robot is not strong enough to carry out the task, gets stuck, and sends a message saying, "Not able to carry out the task.")

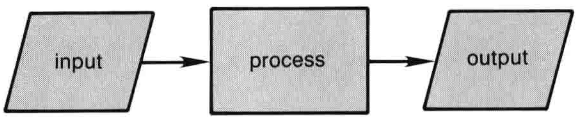
In many ways a computer is similar to that obedient, dumb robot. It must be told each step to carry out. Additionally, each step must be carried out in the correct sequence. The computer is not smart enough to know when steps are out of sequence, so it simply follows directions as long as it possibly can. Sometimes the results will not be what are expected, but you can be sure that the computer followed directions exactly.

Did you instruct the robot to remove the spark plugs with a wrench? Did you tell it to check the gap? Exactly how do you think it will replace the old plugs with the new plugs? If it hasn't been told these things, it will either stop or do something strange and unexpected. In either case, the results can be frustrating.

When writing instructions for the computer to solve a problem, you must be certain to compute the result *before* printing it. Since the computer doesn't know the difference, it will do only and exactly what you ask. Depending on your frame of mind, sometimes the results of computer processing can be uproariously funny or infuriating. Frankly, some people get so frustrated with the level of detail required that the programming would be better left to someone else.

Hardware and Software

In our robot story, you could say that you had a powerful robot (the hardware), but without the proper instructions (the software), the robot would not be much help. The same is true for the computer. The computer itself—the circuitry, the case, the keyboard, the screen—are called the *hardware.* While the computer has tremendous power and flexibility, it absolutely *must* have instructions in order to carry out any useful work. Those instructions are the *software,* or the **computer programs.** Computer programs can be thought of as the instructions necessary to convert inputs into outputs.

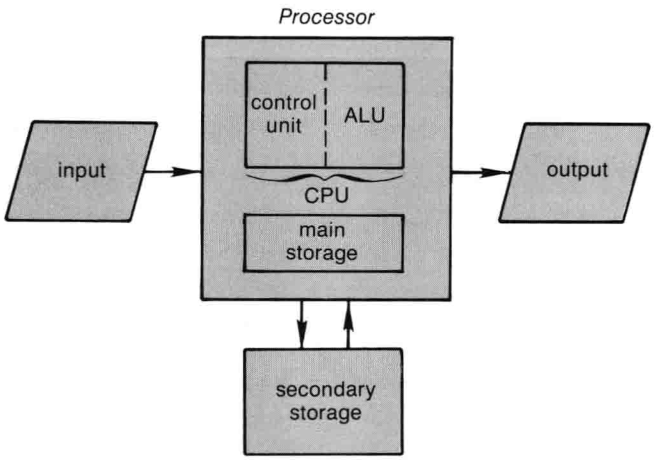


The entire purpose of computer **processing** is to produce some type of **output** (e.g., a report, a computation, a sorted list, a graph). Output may be generated on a screen, a printer, a plotter, a speaker, or some other output device. In order to produce output, there must be processing of the input data. The input to the program may come from the keyboard, from magnetic disk or tape, from a microphone, or from some type of reader such as a bar code reader or card reader. The processing may be arithmetic computations or rearranging, reorganizing, or reformatting data.

The primary goal of this text is to present a method of writing computer software—turning inputs into useful outputs. A basic knowledge of computer hardware is helpful when writing computer instructions.

Hardware

Note the block diagram of a computer. The computer can be seen as a group of components working together. The following sections explain the various components.



Processor

The main unit of the computer is often called the *processor.* In the block diagram, the two primary components of the processor are the *CPU (Central Processing Unit)* and the *main storage* of the computer.

The CPU, sometimes called the "brain" of the computer, does the actual manipulation of the data. One part of the CPU, called the *ALU* (*Arithmetic and Logic Unit*), does all computations and logical operations. Logical operations include such things as determining that one number is greater than another, that one name follows another, or that a condition is *true* or *false.*

Another part of the CPU is the control unit. It is the control unit that controls and coordinates the execution of instructions. The control unit and the ALU working together carry out the instructions of the computer program.

Main Storage

The main storage of the computer can have many different names. The words *storage* and *memory* are used interchangeably when referring to the computer. Main storage may be called *primary storage, internal storage, temporary storage,* or *RAM (Random Access Memory).*

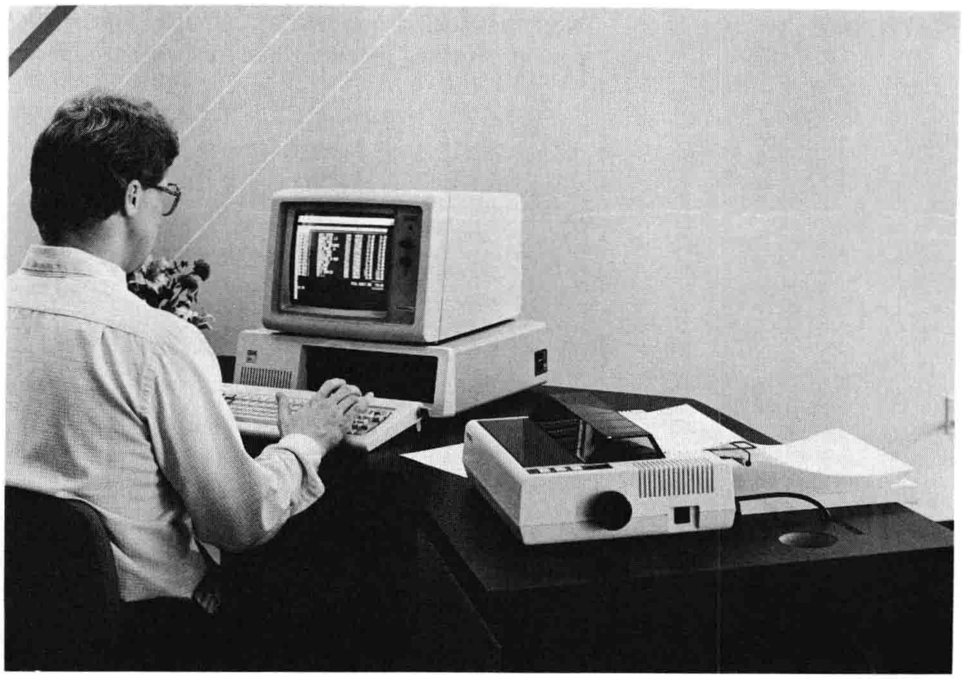
When a computer is executing a program, main storage must hold the entire program as well as some of the data being operated upon. For this reason, the size of main storage is important. In order to hold a large program, a computer must have a sufficient amount of memory.

Computers are sold with varying amounts of main storage, and often additional memory may be added. Memory size is generally stated in terms of the number of characters that can be stored, where one character equals one letter of the alphabet, one digit of a number, or one of the special symbols such as $, %, *.

In computer terms, the amount of storage needed to hold one character is called a **byte.** A group of 1024 bytes is referred to as one **K** (*K*ilobyte). As a rough guide to computer storage, remember that 1 K holds about one thousand characters, which would be approximately two-thirds of a double-spaced, typewritten page of data.

Typical main storage sizes:

48K bytes—approximately 48,000 characters
64K bytes—approximately 64,000 characters
128K bytes—approximately 128,000 characters
256K bytes—approximately 256,000 characters
512K bytes—approximately 512,000 characters
1M—one megabyte—approximately 1 million characters


A personal computer.

Since the entire program and its data must fit into main storage, the computer's memory may be a limiting factor when developing computer programs.

Volatility

As a general rule, you don't need to know how a computer works in order to use one, just as you don't need to understand all of the systems of a car in order to drive. But there are a few pieces of information that will make life easier for you.

One helpful fact to know is that most computer memory is *volatile;* that is, when the power source is removed, the contents of the storage are lost. Any program or data stored in the computer's main storage is gone when the computer is turned off or a power failure occurs. This phenomenon has been the source of many curses and tears by computer users.

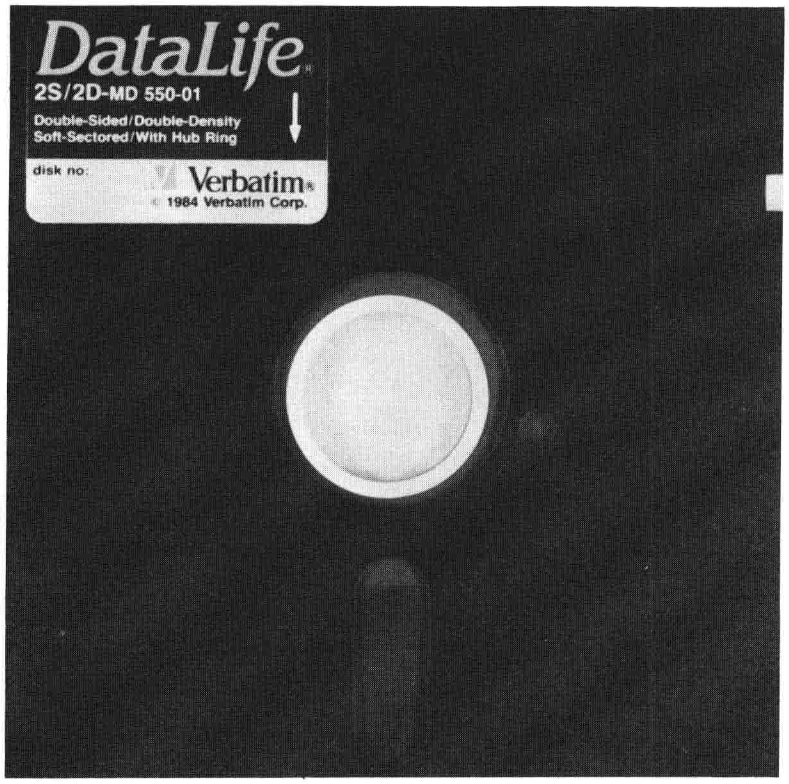
Secondary or Auxiliary Storage

Fortunately, there is storage that is nonvolatile—that does *not* lose its memory when the power goes off. This is the type of memory used for long-term storage of data files and programs—the media used to store programs when they are *not* being executed. This is called *secondary storage, auxiliary storage, external storage* (or memory), or sometimes long-term storage.

The two most common forms of secondary storage are magnetic disk and magnetic tape. Within these two groups, the types are further broken down into small disks, large disks, hard-surface disks, floppy disks, and cassette tapes and reel-to-reel tapes of various sizes and shapes. As a BASIC programmer for a personal computer, the storage media you will most likely see is the **floppy disk,** or **diskette,** which comes in several different sizes. Each computer manufacturer uses a little different method for placing data on a disk. The diskette must be formatted for the computer being used. So, with some exceptions, the general rule is that diskettes for one computer will not be usable on a computer of another manufacturer.

Since it is likely that you will be handling diskettes for storage of programs and data, a few words about the care of disks are in order.

A floppy disk, or diskette.



1. The data is stored magnetically, similar to audio tapes. A magnetic field (magnet, power supply, etc.) can erase or scramble your data.
2. *Anything* on the surface of the disk can ruin the data, and maybe even the read/write heads in the disk drive. This means no fingerprints, hairs, lemonade, or cookie crumbs.
3. Heat ruins disks. Leaving a diskette in a hot car is a sure way to destroy it.
4. Pressure can damage a disk. Never allow a disk to be caught by the ring of a binder. Never use a ballpoint pen to write on the disk label.

Read Only Memory (ROM)

There is another type of nonvolatile memory available for computers. This storage resembles main storage with one important difference—the contents of the memory cannot be changed by the computer user. This *ROM* (*Read Only Memory*) is filled with software by the manufacturer. Once inside the computer, the instructions may be executed, but not changed. ROM is actually hardware containing permanently stored software and is sometimes called *firmware*.

Computer Input

As you have discovered, main storage must hold the program and some of the data to operate upon. How do these things get into storage? Since all data must be stored electronically, letters, numbers, words, or sounds must first be converted to a form that can be stored. Then these data can be moved into main storage. This conversion to electronic form, sometimes called digitizing, takes place in computer input devices.

The foremost input device is the keyboard. What happens when you press the *A* key on the keyboard? That keypress must be converted into digital electronic pulses that can be stored in computer memory.

Many other input devices are available for personal computers as well as their larger counterparts. Increasingly popular as input devices for personal computers are