Schaum's
OUTLINE SERIES

THEORY AND PROBLEMS OF

PROGRAMMING WITH STRUCTURED BASIC

Byron S. Gottfried

- Covers all course fundamentals and supplements any class text
- Teaches effective problem-solving techniques
- 457 Problems and 296 Review Questions
- Contains 265 Programming Problems and 56 Programming Examples
- Includes material on the 1987 ANSI Standard, True BASIC, and Microsoft QuickBASIC/QBASIC

SCHAUM'S OUTLINE OF

THEORY AND PROBLEMS

OF

PROGRAMMING WITH STRUCTURED BASIC

BYRON S. GOTTFRIED, Ph.D.
Professor of Industrial Engineering
University of Pittsburgh

SCHAUM'S OUTLINE SERIES
McGRAW-HILL, INC.

New York St. Louis San Francisco Auckland Bogotá Caracas
Lisbon London Madrid Mexico Milan Montreal
New Delhi Paris San Juan Singapore
Sydney Tokyo Toronto

To Marcia, Sharon, Gail and Susan

BYRON S. GOTTFRIED is a Professor of Industrial Engineering at the University of Pittsburgh. He received his Ph.D. from Case-Western Reserve University in 1962, and has been a member of the Pitt faculty since 1970. His primary interests are in the modeling and simulation of industrial processes. Dr. Gottfried also has active interests in computer graphics and computer programming languages. He is the author of several books, including three editions of *Programming with BASIC*, *Programming with C* and *Programming with Pascal* in the Schaum's Outline Series.

This book is printed on recycled paper containing a minimum of 50% total recycled fiber with 10% postconsumer de-inked fiber.

Schaum's Outline of Theory and Problems of
PROGRAMMING WITH STRUCTURED BASIC

2 3 4 5 6 7 8 9 10 11 12 13 14 15 SH SH 9 8 7 6 5 4 3 2

ISBN 0-07-023899-5

Sponsoring Editor: John Aliano
Production Supervisor: Louise Karam
Editing Supervisors: Meg Tobin, Maureen Walker
Cover design by Amy E. Becker

Gottfried, Byron S., date
Schaum's outline of theory and problems of programming with structured BASIC / Byron S. Gottfried.
p. cm. -- (Schaum's outline series)
Includes index.
ISBN 0-07-023899-5
1. BASIC (Computer program language) 2. Structured programming.
I. Title. II. Title: Outline of theory and problems of programming with structured BASIC. III. Title: Programming with structured BASIC. IV. Series.
QA76.76.B3G68 1993
005.13'3--dc20 91-43808
CIP

Preface

BASIC has been a popular programming language ever since it was first introduced in 1964. Its use has increased explosively, however, as a result of the proliferation of personal computers that occurred during the 1980s. Today, BASIC is the most popular of all programming languages. It is studied in high schools and colleges throughout the world, and remains the language of choice of most computer hobbyists.

As BASIC grew in popularity, it has also matured dramatically in terms of its capabilities. Many of the sophisticated, structured programming constructs that are present in other, newer programming languages have now been added to BASIC. In addition, the graphical user interfaces available on personal computers have brought about a vastly improved programming environment. As a result, BASIC has "grown up." Modern implementations of the language bear little resemblance to the simplistic language of the 1960s.

This book is intended as a replacement for my earlier *Schaum's Outline of Programming with BASIC*. It offers instruction in BASIC programming using the features found in contemporary, structured implementations of the language. Thus, the book stresses the development of programs that are logical, efficient and orderly. The reader is therefore exposed to the principles of good programming practice as well as the specific rules of BASIC.

The book concentrates on three implementations of structured BASIC: the 1987 ANSI standard, True BASIC and Microsoft's QuickBASIC/QBASIC. True BASIC is a popular personal computer implementation that adheres very closely to the 1987 ANSI standard. QuickBASIC, and its twin QBASIC, have personalities of their own. They are widely used on many personal computers.

One of my goals in writing this book is that it be easily understood. This enables the book to be attractive to a wide reader audience, ranging from high school students to practicing professionals. The book is particularly well suited to the advanced secondary or beginning college level, either as a textbook for a beginning programming course, as a supplementary text for a more comprehensive course in analytical techniques or as an effective self-study guide. For the most part, the required mathematical level does not go beyond high school algebra.

The material is organized in such a manner that the reader can write complete, though elementary, BASIC programs as soon as possible. It is very important that the reader write such programs and execute them on a computer concurrently with reading the text. This greatly enhances the beginning programmer's self-confidence and stimulates his or her interest in the subject. (Learning to program a computer is like learning to play the piano; it cannot be learned simply by studying a textbook!)

The text contains many examples. These include both comprehensive programming problems and simple illustrations that focus on specific programming constructs. In addition, sets of review questions, drill problems and programming problems are included at the end of each chapter. The review questions enable readers to test their recall of the material presented within the chapter. They also provide an effective chapter summary. Most of the drill problems and programming problems require no special mathematical or technological background. The student should solve as many of these problems as possible. (Answers to most of the drill problems are provided at the end of the text.) When using this book as a text in a programming course, it may also be advisable for the instructor to supplement the programming problems with additional assignments that reflect particular disciplinary interests.

The principal features of both True BASIC and QuickBASIC/QBASIC are summarized in five appendixes for the reader's convenience. This material should be used frequently for ready reference and quick recall. It will be particularly helpful when writing or debugging a new program.

Finally, readers who complete this book will have learned a great deal about general programming concepts as well as the specific rules of structured BASIC. In addition, they should be convinced that programming with structured BASIC is not only *easy*, but also *fun*.

BYRON S. GOTTFRIED

Complete Programming Examples

The programming examples are listed in the order in which they first appear within the text. The examples vary from very simple to moderately complex. Multiple versions are presented for many of the programs, particularly the simpler programs.

Contents

Chapter 1

Introductory Concepts

BASIC (*B*eginner's *A*ll-purpose *S*ymbolic *I*nstruction *C*ode) is a popular, easily learned programming language that was first developed in the mid-1960s. In recent years BASIC has undergone extensive modifications, in order to provide the same structured programming features that are found in other popular programming languages. These newer versions of BASIC are often referred to as *structured BASIC.*

This book offers instruction in computer programming using the features found in these newer versions of BASIC. By studying this book you will learn the details of writing complete, structured programs in BASIC. The concepts are demonstrated in detail by the many sample problems included within the text.

1.1 INTRODUCTION TO COMPUTERS

Today's computers come in many different forms. They range from massive, multipurpose *mainframes* and *supercomputers* to desktop-size *personal computers.* Between these extremes is a vast middle ground of *minicomputers* and *workstations.* Large minicomputers approach mainframes in computing power, whereas workstations are powerful personal computers.

Mainframes and large minicomputers are used by many businesses, universities, hospitals and government agencies to carry out sophisticated scientific and business calculations. These computers are expensive (large computers can cost millions of dollars) and may require a sizeable staff of supporting personnel and a special, carefully controlled environment.

Personal computers, on the other hand, are small and inexpensive. In fact, portable, battery-powered personal computers smaller than a typewriter are now available. Personal computers are widely used in most schools and businesses and they are rapidly becoming common household items. Students typically use personal computers when learning to program with BASIC.

Figure 1.1 shows a student using a personal computer.

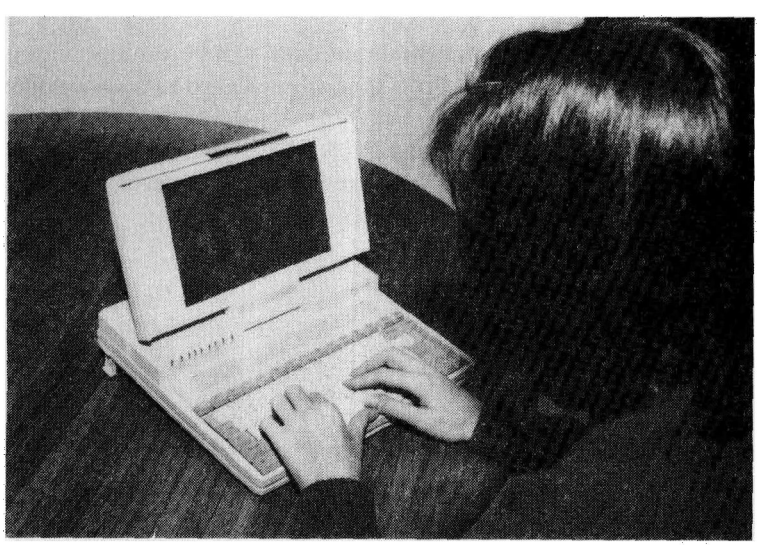

Fig. 1.1

Despite their small size and low cost, modern personal computers approach small minicomputers in computing power. They are now used for many applications that formerly required larger, more expensive computers. Moreover, their performance continues to improve dramatically as their cost continues to drop. The design of a personal computer permits a high level of interaction between the user and the computer. Most applications (e.g., word processors, graphics programs, spreadsheets and database management programs) are specifically designed to take advantage of this feature, thus providing the skilled user with a wide variety of creative tools to write, draw or carry out numerical computations. Applications involving high-resolution graphics are particularly common.

Many organizations connect personal computers to larger computers or to other personal computers, thus permitting their use either as stand-alone devices or as terminals within a computer *network*. Connections over telephone lines are particularly common. When viewed in this context, we see that personal computers often *complement*, rather than *replace*, the use of larger computers.

1.2 COMPUTER CHARACTERISTICS

All digital computers, regardless of their size, are basically electronic devices that can transmit, store, and manipulate *information* (i.e., *data*). Several different types of data can be processed by a computer. These include *numeric data*, *character data* (names, addresses, etc.), *graphic data* (charts, drawings, photographs, etc.), and *sound* (music, speech patterns, etc.). The two most common types, from the standpoint of a beginning programmer, are numeric data and character data. Scientific and technical applications are concerned primarily with numeric data, whereas business applications usually require processing of both numeric and character data.

To process a particular set of data, the computer must be given an appropriate set of instructions called a *program*. These instructions are entered into the computer and then stored in a portion of the computer's *memory*.

A stored program can be *executed* at any time. This causes the following things to happen.

1. A set of information, called the *input data*, will be entered into the computer (from the keyboard, a floppy disk, etc.) and stored in a portion of the computer's memory.

2. The input data will be processed to produce certain desired results, known as the *output data*.

3. The output data, and perhaps some of the input data, will be printed onto a sheet of paper or displayed on a *monitor* (a television receiver specially designed to display computer output).

This three-step procedure can be repeated many times if desired, thus causing a large quantity of data to be processed in rapid sequence. It should be understood, however, that each of these steps, particularly steps 2 and 3, can be lengthy and complicated.

Example 1.1

A computer has been programmed to calculate the area of a circle using the formula $a = \pi r^2$, given a numeric value for the radius r as input data. The following steps are required.

1. Read the numeric value for the radius of the circle.

2. Calculate the value of the area using the above formula. This value will be stored, along with the input data, in the computer's memory.

3. Print (display) the values of the radius and the corresponding area.

Each of these steps will require one or more instructions in a computer program.

The foregoing discussion illustrates two important characteristics of a digital computer: *memory* and *capability to be programmed.* A third important characteristic is its *speed and reliability.* We will say more about memory, speed, and reliability in the next few paragraphs. Programmability will be discussed at length throughout the remainder of this book.

Memory

Every piece of information stored within the computer's memory is encoded as some unique combination of zeros and ones. These zeros and ones are called *bits* (*bi*nary digi*ts*). Each bit is represented by an electronic device that is, in some sense, either "off" (zero) or "on" (one).

Small computers have memories that are organized into 8-bit multiples called *bytes*, as illustrated in Figure 1.2. Notice that the individual bits are numbered, beginning with 0 (for the rightmost bit) and extending to 7 (the leftmost bit). Normally, a single character (e.g., a letter, a single digit or a punctuation symbol) will occupy one byte of memory. An instruction may occupy 1, 2 or 3 bytes. A single numeric quantity may occupy 1 to 8 bytes, depending on its *precision* (i.e., the number of significant figures) and its *type* (integer, floating-point, etc.).

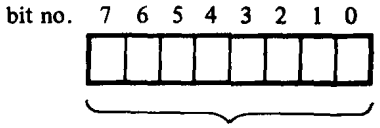

Fig. 1.2

The size of a computer's memory is usually expressed as some multiple of $2^{10} = 1024$ bytes. This is referred to as 1K. Small computers have memories whose sizes typically range from 64K to several megabytes, where 1 megabyte (1M) is equivalent to $2^{10} \times 2^{10} = 1024$K bytes.

Example 1.2

The memory of a small personal computer has a capacity of 256K bytes. Thus, as many as $256 \times 1024 = 262{,}144$ characters and/or instructions can be stored in the computer's memory. If the entire memory is used to represent character data (which is actually quite unlikely), then over 3200 names and addresses can be stored within the computer at any one time, assuming 80 characters for each name and address.

If the memory is used to represent numeric data rather than names and addresses, then over 65,000 individual numbers can be stored at any one time, assuming each numeric quantity requires 4 bytes of memory.

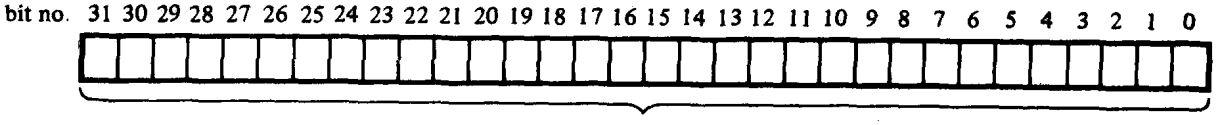

Fig. 1.3

Large computers have memories that are organized into *words* rather than bytes. Each word will consist of a relatively large number of bits - typically 32 or 36. The bit-wise organization of a 32-bit word is illustrated in Figure 1.3. Notice that the bits are numbered, beginning with 0 (for the right-most bit) and extending to 31 (the left-most bit).

Figure 1.4 shows the same 32-bit word organized into 4 consecutive bytes. The bytes are numbered in the same manner as the individual bits, ranging from 0 (for the right-most byte) to 3 (the left-most byte).

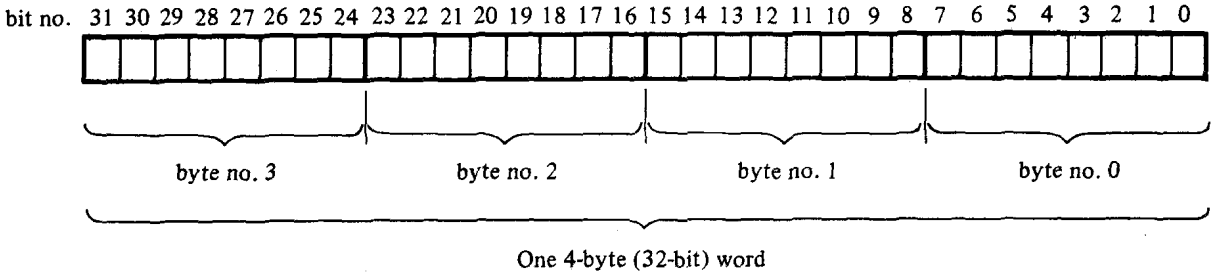

Fig. 1.4

The use of a 32- or a 36-bit word permits one numeric quantity, or a small *group* of characters (typically 4 or 5), to be represented within a single word of memory. Large computers commonly have several million words (i.e., several megawords) of memory.

Example 1.3

The memory of a large computer has a capacity of 2M (2048K) words, which is equivalent to 2048 x 1024 = 2,097,152 words. If the entire memory is used to represent numeric data (which is unlikely), then more than 2 million numbers can be stored within the computer at any one time, assuming each numeric quantity requires one word of memory.

If the memory is used to represent characters rather than numeric data, then about 8 million characters can be stored at any one time, based upon 4 characters per word. This is more than enough memory to store the contents of an entire book.

Most computers also employ *auxiliary storage devices* (e.g., magnetic tapes, disks, optical memory devices) in addition to their primary memories. These devices typically range from 20 or 40 megabytes for a small computer to several hundred megawords for a large computer. Moreover, they allow information to be recorded permanently, since they can often be physically disconnected from the computer and stored when not in use. However, the access time (i.e., the time required to store or retrieve information) is considerably greater for these auxiliary devices than for the computer's primary memory.

Speed and Reliability

Because of its extremely high speed, a computer can carry out calculations within minutes that might require many days, and perhaps even months or years, if carried out by hand. For example, the end-of-semester grades for all students in a large university can typically be processed in just a few minutes on a large computer.

The time required to carry out simple computational tasks, such as adding two numbers, is usually expressed in terms of *microseconds* (1 μsec = 10^{-6} sec) or *nanoseconds* (1 *n*sec = 10^{-3} μsec = 10^{-9} sec). Thus, if a computer can add two numbers in 10 nanoseconds (typical of a modern medium-speed computer), 100 million (10^8) additions will be carried out in one second.

This very high speed is accompanied by an equally high level of reliability. Thus, computers never make mistakes of their own accord. Highly publicized "computer errors," such as a person's receiving a tax refund of several million dollars, are the result of programming errors or data entry errors rather than errors caused by the computer itself.

1.3 MODES OF OPERATION

There are two different ways that a large computer can be shared by many different users. These are the *batch mode* and the *interactive mode*. Each has its own advantages for certain types of problems.

Batch Processing

In *batch processing*, a number of jobs are entered into the computer, stored internally, and then processed sequentially. (A *job* refers to a computer program and its associated sets of input data.) After the job is processed, the output, along with a listing of the computer program, is printed on multiple sheets of paper by a high-speed printer. Typically, the user will pick up the printed output at some convenient time, after the job has been processed.

In *classical batch processing*, the program and the data are recorded on *punched cards*. This information is read into the computer by means of a mechanical card reader and then processed. In the early days of computing all jobs were processed in this manner. Fortunately, this mode of operation is now obsolete.

Modern batch processing is generally tied into a timesharing system (see below). Thus, the program and the data are typed into the computer via a *timesharing terminal* or a personal computer acting as a terminal. The information is then stored within the computer's memory and processed in its proper sequence. This form of batch processing is preferable to classical batch processing, since it eliminates the need for punched cards and allows the input information (program and data) to be edited while it is being entered.

Large quantities of information (both programs and data) can be transmitted into and out of the computer very quickly in batch processing. Furthermore, the user need not be present while the job is being processed. Therefore, this mode of operation is well-suited to jobs that require large amounts of computer time or are physically lengthy. On the other hand, the total time required for a job to be processed in this manner may vary from several minutes to several hours, even though the job may have required only a second or two of actual computer time. (Each job must wait its turn before it can be read, processed, and printed out.) Thus, batch processing is undesirable when processing small, simple jobs that must be returned as quickly as possible (as, for example, when learning computer programming).

Timesharing

Timesharing allows many different users to use a single computer simultaneously. Generally, the host computer is a mainframe or a large minicomputer. The various users communicate with the computer through their own individual terminals. In a modern timesharing network, personal computers are often used as timesharing terminals. Since the host computer operates much faster than a human sitting at a terminal, one large computer can support many terminals at essentially the same time. Therefore, each user will be unaware of the presence of any other users, and will seem to have the host computer at his or her own disposal.

An individual timesharing terminal may be wired directly to the host computer, or it may be connected to the computer over telephone lines, a microwave circuit, or even an earth satellite. Thus, the terminal can be located far — perhaps hundreds of miles — from its host computer. Systems in which personal computers are connected to large mainframes over telephone lines are particularly

common. Such systems make use of *modems* (i.e., *mo*dulator/*demo*dulator devices) to convert the digitized computer signals into analog telephone signals and vice versa. Through such an arrangement a person working at home, on his or her own personal computer, can easily access a remote computer at school or at the office.

Timesharing is best suited for processing relatively simple jobs that do not require extensive data transmission or large amounts of computer time. Many applications that arise in schools and commercial offices have these characteristics. Such applications can be processed quickly, easily, and at minimum expense using timesharing.

Example 1.4

A major university has a computer timesharing capability consisting of 200 hard-wired timesharing terminals and 80 additional telephone connections. The timesharing terminals are located at various places around the campus and are wired directly to a large mainframe computer. Each terminal is able to transmit information to or from the central computer at a maximum speed of 960 characters per second.

The telephone connections allow students who are not on campus to connect their personal computers to the central computer. Each personal computer can transmit data to or from the central computer at a maximum speed of 240 characters per second. Thus, all 280 terminals and personal computers can interact with the central computer at the same time, though each student will be unaware that others are simultaneously sharing the computer.

Interactive Computing

Interactive computing is a type of computing environment that originated with commercial timesharing systems and has been refined by the widespread use of personal computers. In an interactive computing environment, the user and the computer interact with each other during the computational session. Thus, the user may periodically be asked to provide certain information that will determine what subsequent actions are to be taken by the computer and vice versa.

Example 1.5

A student wishes to use a personal computer to calculate the radius of a circle whose area has a value of 100. A program is available that will calculate the area of a circle, given the radius. (Note that this is just the opposite of what the student wishes to do.) This program isn't exactly what is needed, but it does allow the student to obtain an answer by trial and error. The procedure will be to guess a value for the radius and then calculate a corresponding area. This trial-and-error procedure continues until the student has found a value for the radius that yields an area sufficiently close to 100.

Once the program execution begins, the message

```
Radius = ?
```

is displayed. The student then enters a value for the radius. Let us assume that the student enters a value of 5 for the radius. The computer will respond by displaying

```
Area = 78.5398

Do you wish to repeat the calculation?
```

The student then types either `yes` or `no`. If the student types `yes`, the message

```
Radius = ?
```

again appears, and the entire procedure is repeated. If the student types no, the message

```
Goodbye
```

is displayed and the computation is terminated.

Shown below is a printed copy of the information displayed during a typical interactive session using the program described above. In this session, an approximate value of $r = 5.6$ was determined after only three calculations. The information typed by the student is underlined.

```
Radius = ? 5
Area = 78.5398

Do you wish to repeat the calculation? yes

Radius = ? 6
Area = 113.097

Do you wish to repeat the calculation? yes

Radius = ? 5.6
Area = 98.5204

Do you wish to repeat the calculation? no

Goodbye
```

Notice the manner in which the student and the computer appear to be conversing with one another. Also, note that the student waits until he or she sees the calculated value of the area before deciding whether or not to carry out another calculation. If another calculation is initiated, the new value for the radius supplied by the student will depend on the previously calculated results.

Programs designed for interactive computing environments are sometimes said to be *conversational* in nature. Computerized games are excellent examples of such interactive applications. This includes fast-action, graphical arcade games, even though the user's responses may be reflexive rather than numeric or verbal.

1.4 TYPES OF PROGRAMMING LANGUAGES

Many different languages can be used to program a computer. The most basic of these is *machine language* — a collection of very detailed, cryptic instructions that control the computer's internal circuitry. This is the natural dialect of the computer. Very few computer programs are actually written in machine language, however, for two significant reasons: first, because machine language is very cumbersome to work with; and second, because every different type of computer has its own unique instruction set. Thus, a machine-language program written for one type of computer cannot be run on a different type of computer without significant alterations.

Usually, a computer program will be written in some *high-level* language, whose instruction set is more compatible with human languages and human thought processes. Most of these are *general-purpose* languages such as BASIC, C, Pascal and Fortran. There are also various *special-purpose* languages whose instruction sets are specifically designed for some particular type of application. Some common

examples are LISP, a *list-processing* language that is widely used for artificial intelligence applications, and CSMP and SIMAN, two different types of special-purpose *simulation* languages.

As a rule, a single instruction in a high-level language will be equivalent to several instructions in machine language. This greatly simplifies the task of writing complete, correct programs. Furthermore, the rules for programming in a particular high-level language are much the same for all computers, so that a program written for one computer can generally be run on many different computers with little or no alteration. Thus, we see that a high-level language offers three significant advantages over machine language: *simplicity, uniformity* and *portability* (i.e., machine independence).

A program that is written in a high-level language must, however, be translated into machine language before it can be executed. This is known as *compilation* or *interpretation*, depending on how it is carried out. (Compilers translate the entire program into machine language before executing any of the instructions. Interpreters, on the other hand, proceed through a program by translating and then executing single instructions or small groups of instructions.) In either case, the translation is carried out automatically within the computer. In fact, inexperienced programmers may not even be aware that this process is taking place, since they typically see only their original high-level program, the input data, and the calculated results. Most implementations of BASIC operate as interpreters, though compilers are becoming increasingly common.

A compiler or interpreter is itself a computer program. It accepts a program written in a high-level language (e.g., BASIC) as input, and generates a corresponding machine-language program as output. The original high-level program is called the *source* program, and the resulting machine-language program is called the *object* program. Every computer must have its own compiler or interpreter for a particular high-level language.

It is generally more convenient to develop a new program using an interpreter rather than a compiler. Once an error-free program has been developed, however, a compiled version will normally execute much faster than an interpreted version. The reasons for this are beyond the scope of our present discussion.

1.5 INTRODUCTION TO BASIC

BASIC is a general-purpose, easy-to-use programming language. Its instructions consist of terms that resemble algebraic expressions, augmented by certain English keywords such as IF, THEN, FOR, DO, SELECT, INPUT, and PRINT. Other high-level languages have similar features, though they tend to be more complicated to use. Hence, BASIC is particularly well-suited for persons learning to program for the first time. In fact, most high schools and many junior high schools now provide instruction in BASIC programming.

The use of BASIC is by no means restricted, however, to elementary programming exercises. It is often used for more advanced applications in business, science, engineering and mathematics. Moreover, BASIC is the principal language that is used with personal computers (PCs). Thus, BASIC is used for many novel applications, such as computer games that require the use of graphics and sound enhancements. We will see a representative sampling of these different types of programming applications in the examples included within this book.

History of BASIC

BASIC was originally developed in 1964 at Dartmouth College by John Kemeny and Thomas Kurtz. At that time all computing was carried out on mainframe computers using batch processing. Fortran, Algol and COBOL were the standard programming languages. The original BASIC was much easier to use than any of these languages, however, and it could be run interactively in a timesharing environment. Thus, BASIC provided a radical improvement in the way computing was carried out at that time.