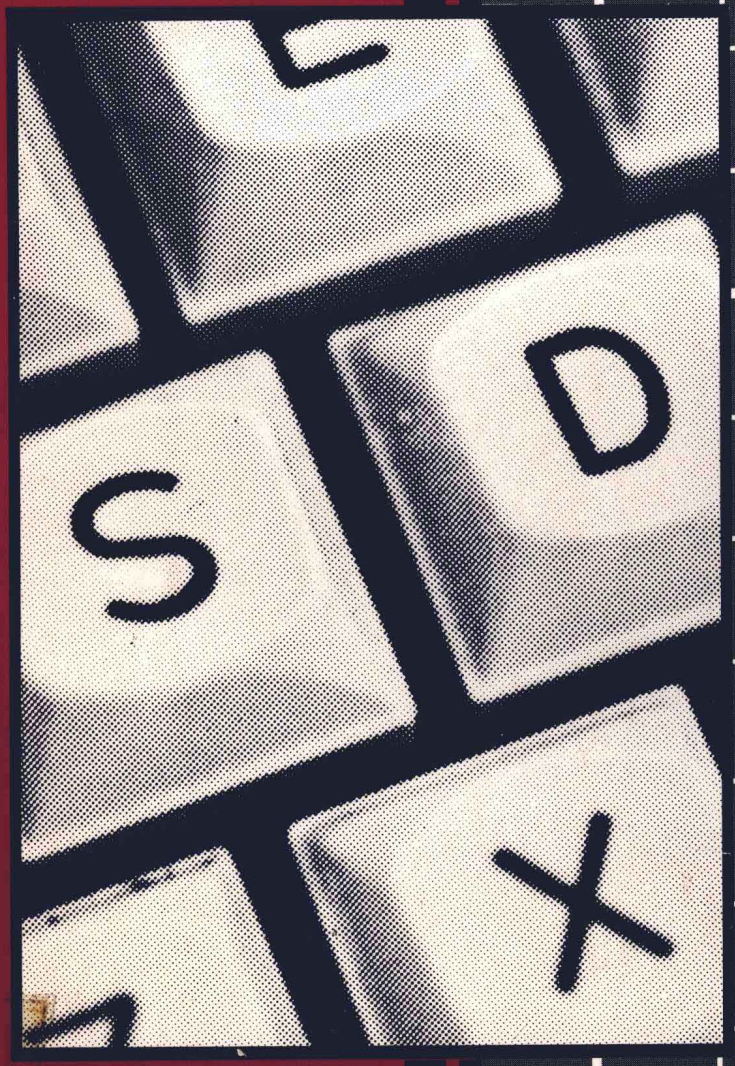
INTRODUCTION TO STRUCTURED PROGRAMMING USING BASIC



COLEMAN
BARNETT

INTRODUCTION TO STRUCTURED PROGRAMMING USING BASIC

Coleman Barnett
Tarrant County Junior College

Gorsuch Scarisbrick Publishers
Dubuque, Iowa 52001

Gorsuch Scarisbrick Publishers
576 Central
Dubuque, IA 52001

10 9 8 7 6 5 4 3 2 1

ISBN 0-89787-402-1

Printed in the United States of America

Preface

The overwhelming growth in the use of computers in our society has created an increasing need for skilled computer programmers. As the complexity of the field increases, the need is especially great for programmers well-versed in the concepts and techniques common to *all* programming languages rather than in the techniques utilized by any single language. Learning a specific programming language does not necessarily ensure learning how to prepare good computer programs.

This text is designed to help students learn how to write structured computer programs. Emphasis is placed on learning the concepts and techniques of algorithm development that form the basis for writing good structured programs. The reader will then be able to transfer this knowledge into programming languages and environments that require structured techniques. BASIC has been chosen to demonstrate the programming applications of these concepts because of its widespread use, its use in programming classes, and its relatively easier syntax and semantics.

The text follows a parallel chapter organization through many of the chapters in order to demonstrate and reinforce the concepts and techniques to be learned. Algorithm development in both flowchart and pseudocode form are shown in a given chapter. The corresponding BASIC language program development for these algorithms is in the following chapter. This allows the student to become familiar with the algorithm development independent of programming language concerns. The following chapter then demonstrates the BASIC language requirements for the implementation of the algorithms.

The early chapters are introductory chapters on problem solving, the nature of an algorithm, and computer and data organization. Chapters 4 and 5 are parallel chapters on getting started, with the former covering in some detail the techniques of flowcharting and pseudocode, and the latter covering program development in general and BASIC programming specifically. Chapters 6 and 7 develop in a nonstructured approach the logic of sequence, loops, and selection. Parallel chapters 8 and 9 place the logic structures learned in chapters 6 and 7 in a structured programming framework. This allows the student to put in perspective the differences between structured and nonstructured programming. Chapter 10 introduces automatic loop control (FOR/NEXT). Next, the parallel chapters 11 and 12 present the use of one-dimensional arrays in processing. Included is the use of arrays for sorting and indexing. As an extension of these chapters, chapter 13 then discusses nested loops and two-dimensional arrays. The next two chapters are parallel chapters that develop sequential file processing. They are followed by chapters on formatting output using TAB and USING with PRINT, suggestions on debugging, and number and coding systems. These three chapters are treated independently and can be studied at any point in the course.

The text develops structured programming concepts and techniques using common BASIC statements. The solutions to problems in the chapters are explained step by step. Also, the problem solutions in the chapters include walkthroughs that show the results of running data through the algorithms

step by step in the flowcharts and pseudocode and line by line in the computer programs. Each chapter closes with a short summary, a list of key terms, and several review questions designed to help students test their learning. In addition, the exercises at the end of each chapter contain problems that are simpler than chapter examples, problems with the same degree of difficulty, and problems that are more difficult than chapter examples. This enables students to check their ability to apply the concepts and techniques learned in each chapter.

I appreciate the efforts of many people in developing this book—in particular the following reviewers: Mike Michaelson, Associate Professor, Computer Information Systems Department, Palomar College; Leona Roen, Associate Professor, Data Processing Department, Oakton Community College; and Michael Walton, Associate Professor, Business Data Processing Department, Miami Dade Community College, North Campus. Their comments and suggestions were especially helpful in producing a complete and accurate text.

Coleman Barnett

Contents

1 Problem Solving

WHAT IS A PROBLEM?

This may seem to be a silly question. After all, everyone knows a problem is something that has to be solved. Is there a person alive who has not ever stated, "I have a problem"—or at times, "you have a problem"?! Webster's dictionary states what may be obvious; that is, that a problem is "a question proposed for solution or consideration."

This leads us to a second question: why would a computer programmer be concerned about what a problem is? The answer is because many times a person does not see the relationship between a problem and the procedure needed to solve it. Computer programming is a problem-solving procedure. Learning about computers is without much merit unless the knowledge can be used to help solve some problem. The procedure used to solve a problem—programming— is of primary importance. Using a computer is of secondary importance.

A problem that might be solved by a computer programmer could involve such concerns as payrolls, general ledgers, sales analysis, depreciation, amortization schedules, accounts payable, accounts receivable, production schedules, or many other business-related record-keeping problems. A programmer working in these areas would be a business or commercial programmer.

Another group of problems a computer programmer might help solve include calculating the area of a circle, determining a space vehicle's flight path, or figuring the length of the third side of a right triangle. A programmer working on these mathematical and science-related problems would be a scientific programmer. As the usefulness of the computer increases, programmers will deal with problems in many other fields as well—for example, in medicine, in government, and even in music.

Since a computer program is a problem-solving procedure, the first priority of anyone writing a computer program is to determine the problem. Although this may seem obvious, it is not unusual for beginning programming students to get so involved with procedures that they neglect to understand the problems. Understanding a problem involves more than knowing that payroll calculates pay. It involves knowing or learning how the pay is calculated. Remember, if a problem is to be solved, the problem must first be understood. Only then can we develop a procedure that will solve the problem.

PROBLEM SOLVING: AN ACQUIRED SKILL

Problem solving involves understanding what the problem requires and then developing procedures to meet these requirements. One school of thought holds that a person either has or doesn't have problem-solving ability; that is, if a person is not born with the ability to solve problems, he can never acquire it. One fallacy of this type of reasoning is that one can't determine for sure whether he has the ability to solve problems until he has solved

at least one. A more positive approach might be to assume that problem solving is a skill that can be developed or even acquired (learned) as other skills are.

To develop or learn problem-solving skills, one can either begin by studying some problem-solving techniques or by actually solving problems. In this text we'll concentrate on a formal method of study that helps develop these skills, as well as on exercises that provide practical experience in problem solving.

COMPUTER-RELATED PROBLEMS

All the problems discussed in this book allow the use of computers for their solutions. Not all problems allow the use of a computer. There are three fairly common characteristics of problems that make it feasible to use a computer: they are repetitive, they are numerical, and they are definable.

Repetitive

When a problem is of a repetitive nature, it has to be solved over and over. An example would be the problem of figuring and distributing payroll. This is a problem that requires a solution weekly or perhaps monthly, depending on the pay period an organization uses. The reason that repetition is important in using the computer to solve problems is because the procedure (or the solution) can be used over and over. The cost of developing a computer procedure to solve a problem is expensive. For example, the process that computerizes payroll procedures for an organization may cost $100,000 to develop and save only $2,000 each pay period. If the problem did not recur, the organization would lose the $98,000 cost that could not be recovered. But if the problem recurs fifty times (if the pay period is weekly), $2,000 times fifty would be $100,000. The time to recover the $100,000 spent on computerizing payroll would thus be fifty weeks. Each pay period after the fifty-week period would result in a savings of $2,000.

Numerical

First and foremost, the computer is a number-manipulating machine. A problem has to be definable quantitatively, or numerically, to be solved by using a computer. When a baby cries, a problem is causing the crying. But this problem cannot be reduced to a defined numerical equation. Probably the child needs understanding, care, love, and compassion. The computer is short of these qualities—at least as far as we know!

Definable

A problem that cannot be defined or explained probably cannot be solved. If a workable solution occurs without problem definition, someone has been lucky. It would be virtually impossible to develop procedures to solve a given problem by computer if the problem has not been defined. We'll look at problem definition in more detail later.

WHAT IS DATA PROCESSING?

Data processing is the manipulation of data. Data includes information such as name, social security number, address, employee identification number, pay rate, sales amount, discount, account number, invoice number,

amount of depreciation, or any of the many other items of information that are generated by a business and that need to be processed. Data processing involves collecting, organizing, and processing this data.

By the time data is processed, it is usually referred to as information. In a general sense, data processing and information processing are the same. In a specific sense, however, *data* refers to one *item*, such as name, hours worked, or pay rate. *Information* is a more modern or progressive term that refers to a meaningful *relationship* between items of data. A *sales analysis* is a collection of different pieces of data, but collectively, it is information useful to those who understand the relationships between the figures.

Organizing different kinds of data into information is the central goal or purpose of data processing. Within this framework, then, many problems occur that require *procedures* to solve them. Computer programming is a problem-solving procedure. To use the computer to solve a problem, a person must be able to organize data and define the solution in procedural terms that the computer can "understand," or that are compatible with the computer's capabilities.

STEPS IN PROBLEM SOLVING

Any effort made toward solving a problem accomplishes more if the work follows a schedule or plan. This is true whether the problem is analyzing a biological specimen, building a bridge, teaching a child how to read, determining the illness of a patient, or writing a computer program.

Problem solving in a numerical-processing environment involves four steps. They are:

1. Input
2. Processing } Considered the *process* in a three-step
3. Decision } procedure of Input-Process-Output
4. Output

Almost any procedure developed will involve these four steps.

A close observation of any procedure will show that the complexity of the solution is caused by multiple inputs, many kinds of processing, a variety of decisions, and multiple output formats. All of these are interrelated in the procedure. The key to understanding the procedure is to know exactly what it is doing and to be sure the correct procedure is being used. Using the correct procedure means using the correct combination of inputs, processes, decisions, and outputs.

There are nine steps that could be performed in solving a problem by computer. The list of steps is not inclusive or exclusive; that is, it might be made either shorter or longer. The list is sufficiently comprehensive to give one insight into what needs to be done to solve a problem. The steps are normally done in the order listed.

1. *Define the problem.* Computer programming is a means to an end, and the end is the solution to a problem. Any problem to be solved must be defined, or explained; otherwise, not much progress will be made in solving it. Problem definition involves recognizing that there is a problem and explaining what has to be done to solve it.

2. *Plan the required output.* Output must be planned early in the overall procedure because the desired output affects the procedures to be used in solving the problem. If one takes the approach that whatever results are produced as output will be used, the procedures planned may not produce the desired output and the problem will remain unsolved.

As an example, one type of output required in managing payroll would be checks containing earnings for employees. A programmer that causes the computer to write $25.00 rather than $250.00 on a check would need more work on planning correct output.

3. *Specify the input needed to produce the planned output.* After planning the output, the next step is to determine the needed input. The items of data to be processed constitute input. The desired output determines the needed input; that is, if earnings are to be the output, then hours worked and pay rate will be needed as input. Other input items for earnings would include social security and income tax rates for calculating the amount to withhold from gross pay. Obviously, there is no need to think about input before determining output.

4. *Devise a procedure to obtain output from input.* Once the outputs and inputs are determined, a procedure is required to process the input data into the output data. This procedure would normally involve decisions and calculations. In a payroll problem, the procedure would calculate net pay as output by multiplying the input data of hours worked by pay rate and subtracting any deductions. If an employee is paid overtime at twice the regular rate for working over forty hours this calculation would also have to be part of the procedure.

Such a planned procedure is called an *algorithm.* Three structures that are used to develop algorithms are flowcharts, pseudocode, and computer programs; these techniques of algorithm development are covered in subsequent chapters. These three methods are not the only structures that can be used to develop algorithms, but they are used extensively. Flowcharting and pseudocode are techniques used to develop a procedure. A computer program, of course, is the final algorithm or set of instructions given to the computer.

5. *Determine the data to be retained during processing.* As the procedure is developed, certain data that are generated at one point in the procedure may be needed again at a later point. It is the responsibility of the one preparing the procedure to make sure such data is in fact retained for future use. For example, once the income tax deduction is calculated in a payroll problem, the amount must be retained until all deductions are calculated, at which time all of them can be added together to calculate total deductions.

6. *Consider alternate processing possibilities.* Although at least one procedure must be developed to solve a problem, most of the time there are other procedures that will also work. The problem solver, or computer programmer, must be flexible in developing alternative procedures. The problem solver should have or develop the ability to recognize different ways to solve a problem and must be able to discern which is the best way.

In a payroll problem one procedure to calculate net pay could be to:

1. Calculate regular pay
2. Calculate overtime pay
3. Add regular to overtime pay to get gross pay
4. Calculate income tax deduction
5. Subtract income tax deduction from gross pay to get an intermediate result
6. Calculate social security deduction
7. Subtract social security deduction from the intermediate result to get net pay

A better procedure for the same problem might be to:

1. Calculate regular pay
2. Calculate overtime pay

3. Add regular to overtime pay to get gross pay
4. Calculate income tax deduction
5. Calculate social security deduction
6. Add income tax deduction to social security deduction to get total deductions
7. Subtract total deductions from gross pay to get net pay

The second approach calculates total pay and then total deductions before subtracting to arrive at net pay. The first approach calculated gross pay and subtracted each deduction as it was generated to arrive at an intermediate result.

It should be noted that if the problem definition calls for an employee's pay to be generated after each deduction calculation, then the first procedure would have to be used. It is important to use the best procedure possible based on the problem definition.

7. *Write the computer program.* After the procedure has been planned, a computer program can be written using a programming language. The program will consist of instructions to the computer to perform certain tasks. In learning a programming language it is important to know what each instruction will do so they can be put in the correct sequence for the computer to follow.

8. *Test and debug the program.* After the program has been prepared, the computer will follow each of the instructions in the sequence they have been written in. In computer terminology it is said that the computer *executes* the program. If all instructions are written correctly and in the correct order, the input data will produce the correct output. Testing the program means making sure the correct output is produced from the input data.

Even the most experienced programmer seldom writes or prepares a program that works correctly the first time. In testing the program, errors will show up and corrections will be made by the programmer. These corrections may involve changing the form of an instruction, changing the place the instruction falls in the program, removing instructions from the program, or adding new instructions to the program. After corrections are made to the program it is tested again with input data. If the output data is correct then the program is ready to use; if not, the program requires more changes and more testing. This procedure of testing and correcting continues until the program produces correct output. Errors in a computer program are called bugs and correcting these errors is called debugging.

9. *Document the program.* Documentation is any written material that helps explain the nature of a computer program. The amount and type of documentation required depends on the teacher in a classroom situation or on the organization if one is a practicing programmer.

A computer program, especially a large one, is very detailed and may include many intricate relationships between the various instructions. For this reason, written explanation is sometimes necessary for programmers to understand the program. This is especially true when the program may have to be changed in the future. A programmer who writes a program sometimes forgets exactly how a set of instructions process the input data.

Documentation may take many forms. Some of the more common are:

1. A written narrative description of what the program is doing and how this is accomplished
2. A flowchart and/or pseudocode
3. Comments within the program explaining what various instructions or groups of instructions are accomplishing

4. Input descriptions
5. Output descriptions
6. Descriptions of any special processing techniques

SUMMARY

This chapter has introduced the idea of a problem. Computer programming involves problem solving.

"What is a problem?" is more than a tricky question. It is important that a person develop the ability to recognize problems as the first step to solving them. Problem solving techniques should be viewed as something to be learned and practiced.

The world of problem solving is virtually endless. Not all problems are candidates for computer solutions. Those that are, however, form a very sizable group. The problems that tend to be good candidates for computer solution have the three common characteristics of being repetitive, numerical, and definable or explainable.

Data processing is more than merely processing pieces of data. The more inclusive term "information processing" suggests that what is processed should be meaningful and have a justifiable reason for processing; it should be useful information.

Any problem to be solved should be approached in a systematic manner. The basic steps for problem solving include:

1. Defining the problem
2. Planning the required output
3. Specifying the input needed to produce the planned output
4. Devising a procedure to obtain the output from the input
5. Determining the data to be retained during processing
6. Considering alternative procedure possibilities
7. Writing the computer program
8. Testing and debugging the program
9. Documenting the program

TERMS

algorithm
bug
business programmer
data processing
debug
decision
execute
input
output
problem
problem solving
process
scientific programmer
testing

REVIEW QUESTIONS

1. Name the two general categories of computer programmers and explain the work of each.
2. Name three problems that are normally considered to be business related.
3. Name three problems that might not be candidates for computer solution and note why.
4. Differentiate between data processing and information processing.
5. Name the four basic functions needed for problem solving in a computer environment. These four functions are sometimes reduced to what three basic functions?
6. Name the steps that supply a systematic approach to problem solving.