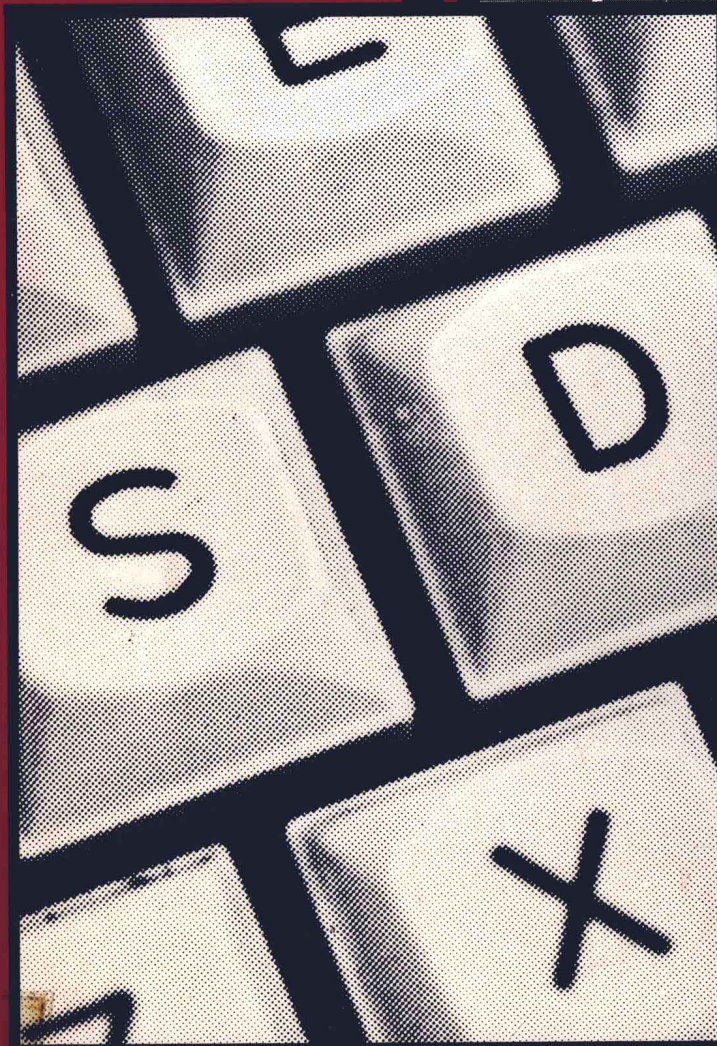


INTRODUCTION TO **STRUCTURED** **PROGRAMMING** USING BASIC



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

Copyright © 1984 by Gorsuch Scarisbrick Publishers

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopy, recording or otherwise, without the prior written permission of Gorsuch Scarisbrick Publishers.

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

Preface xiii

1 Problem Solving 1

What Is a Problem? 1

Problem Solving: An Acquired Skill 1

Computer-Related Problems 2

Repetitive 2

Numerical 2

Definable 2

What Is Data Processing? 2

Steps in Problem Solving 3

Summary 6

Terms 6

Review Questions 6

Exercises 7

2 Algorithms 9

Definition of an Algorithm 9

Characteristics of an Algorithm 9

Precision 9

Finiteness 9

Effectiveness 9

Input 10

Output 10

The Terminology of Algorithms 10

Input, Process, and Output 10

Variable 10

Assignment 11

Expression 11

Statement 13

The Development of an Algorithm 13

The Flowchart as an Algorithm 14

The Pseudocode as an Algorithm 15

The Computer Program as an Algorithm 15

Summary 16

Terms 16

Review Questions 16

Exercises 16

3 Computer and Data Organization 17

Computer Units 17

The Computer System 17

Memory 19

Control 19

Arithmetic/Logic 19

Computer Units versus Steps in Algorithms 20

Data Recording Media and Devices 20

The Data Hierarchy 20

Magnetic Tape 21

Video Display/Keyboard 22

Printer 23

Summary 23

Terms 25

Review Questions 25

Exercises 25

4 Algorithm Development: Getting Started 27

Flowcharts 27

Symbols and Flowchart

Structures 27

The Computer's Thinking Ability 35

Flowcharting Guidelines 37

Other Useful Information on

Flowcharting 42

Pseudocode 43

- Analysis of Pseudocode in Figure 4.6 43
- Analysis of Pseudocode in Figure 4.7 45
- Analysis of Pseudocode in Figure 4.8 46

Summary 47**Terms 47****Review Questions 48****Exercises 48****5 Computer Programming: Getting Started 49****Programming Requires****Precision 49****Categories of Instructions 49****Computer Programming****Concepts 50**

- Programming Proficiency 50
- Algorithm Development: Preparing for the Program 51
- Choosing the Best Computer Programs 52
- The Programmer's Responsibility 52

Converting the Solution of a Problem to a Computer**Program 52**

- Walkthrough 53

Debugging 54**Programming Languages 54****Compiler versus Interpretive****Programming Languages 55**

- Analysis of a Compile Procedure 55
- Analysis of an Interpretive Procedure 56
- Interpretive versus Compiler Languages 58

The BASIC Language 59

- Syntax Rules 59
- Program Modification 63
- Systems Commands 65

Summary 66**Terms 67****Review Questions 67****Exercises 67****6 Algorithm Development: A Way of Thinking 69****Logical Structures 69**

- Sequence 69
- Selection 70
- Loops 72

The Accumulation Process 82**An Example Problem (Problem****6.1) 83**

- Analysis of Flowchart Solution for Problem 6.1 84
- Pseudocode Solution for Problem 6.1 86

Logic Pattern Combinations 87**A Second Example Problem (Problem****6.2) 88**

- Analysis of Flowchart Solution for Problem 6.2 88
- Pseudocode Solution for Problem 6.2 89

Summary 91**Terms 91****Review Questions 91****Exercises 92**

- Sequence 92
- Selection 92
- Loops 92
- General 93

7 Computer Programming: Developing a Way of Thinking 95**Types of Statements 95****Logical Structures 95**

- Sequence Logic 96

READ and DATA Statements 96

- Rules for Using a READ Statement 97
- Rules for Using a DATA Statement 97
- Examples of READ/DATA Statements 98

PRINT Statements 100**LPRINT Statements 101****PRINT Statements with Items to Print Separated by Commas 101**

PRINT Statements with Semicolons	104
LET Statements	106
Summary of the LET Statement	107
END Statements	108
REMARK Statements	108
Sequence Logic with READ, DATA, PRINT, and LET Statements	108
Analysis of the Program in Figure 7.5	109
GOTO Statements	110
IF ... THEN Statements	110
Relational Symbols in BASIC	111
Selection with IF/THEN Statements	111
Analysis of the Program in Figure 7.7	112
The Physical Form of Logic on Flowcharts versus the Physical Form of Logic in Computer Programs	114
Loops	115
Counter-Controlled Loops	115
End-of-File Controlled Loops	123
Other Conditions Controlling Loops	124
The Accumulation Process	127
Walkthrough of the Program in Figure 7.15	128
Problem 7.1	128
Walkthrough of the Program in Figure 7.16	130
Walkthrough of the Program in Figure 7.16 Showing Output for Each Loop	130
Problem 7.2	131
Solution to Problem 7.2	131
Summary	134
BASIC Statements	135
Terms	135
Review Questions	135
Exercises	135
Sequence	136
Selection	136
Loops	136
General	137

8 Algorithm Development: Structured Design as a Way of Thinking 139

Why Structured?	139
Logic Control Structures	140
Sequence	141
IF-THEN-ELSE	141
DO-UNTIL and DO-WHILE	143
Structured Design	145
Top-Down Design and the Control Module	149
Problem 8.1	153
Flowchart Solution to Problem 8.1	153
Pseudocode Solution to Problem 8.1	157
Problem 8.2	158
Analysis of Problem 8.2	159
Problem 8.3	169
Analysis of Problem 8.3	169
Summary	176
Terms	177
Review Questions	177
Exercises	177
Sequence	177
IF-THEN-ELSE	177
DO-UNTIL; DO-WHILE	178
General	178

9 Computer Programming: Structured Design as a Way of Thinking 181

ELSE Statements for Logic-Control Structures	181
Logic-Control Structures	182
Sequence	182
IF-THEN-ELSE	182
DO-UNTIL	185
DO-WHILE	187
GOSUB, RETURN: Statements for Structured-Program Organization	189
Structured-Program Organization	192
Analysis of the Program in Figure 9.6	193

Analysis of the Program in Figure 9.7	193	Analysis of the Flowchart in Figure 11.4	246
Problem 9.1	196	Referencing Only Needed Items	250
Analysis of Problem 9.1	196	Building an Array	250
Solution to Problem 9.1	197	Reading Data into an Array	251
Problem 9.2	201	Generating Data to Place into an Array	254
Analysis of Problem 9.2	201	Guidelines for Using Arrays	258
Solution to Problem 9.2	201	Sorting Arrays	258
Problem 9.3	210	Analysis of the Flowchart in Figure 11.12	260
Analysis of Problem 9.3	210	Pseudocode for the Flowchart in Figure 11.12	262
Solution to Problem 9.3	211	Switches	262
Summary	217	Problem 11.1	263
BASIC Statements	217	Analysis of Problem 11.1	263
Terms	217	Solution to Problem 11.1	263
Review Questions	217	Problem 11.2	267
Exercises	218	Analysis of Problem 11.2	269
Sequence	218	Solution to Problem 11.2	271
IF-THEN-ELSE	218	Summary	276
DO-UNTIL and DO-WHILE	219	Terms	277
General	219	Review Questions	277
10 Automatic Loop Control	223	Exercises	277
Automatic Loop Control	223	12 Computer Programming: Array Processing	279
Flowcharts	223	What Is an Array?	279
Pseudocode	226	Why Use Arrays?	279
Computer Programming	227	Building an Array	280
FOR/NEXT Statements	227	Rules on the Use of the DIM Statement	280
STEP Statement	229	Array Processing: An Example Problem	280
Decrementing at NEXT	230	Generating Data in the Algorithm to Place in an Array	286
The Flexibility of FOR/NEXT	231	Guidelines for Using Arrays	287
FOR/NEXT in Structured Design	232	Switches	288
Automatic Versus Nonautomatic Loop Control	232	Sorting Arrays	290
Summary	236	First-Sort Technique	290
BASIC Statements	236	Second-Sort Procedure	293
Terms	236	Problem 12.1	295
Review Questions	236	Analysis of Problem 12.1	295
Exercises	237	Solution to Problem 12.1	296
11 Algorithm Development: Array Processing	239	Problem 12.2	299
What Is an Array?	239	Analysis of Problem 12.2	299
Why Use Arrays?	239	Solution to Problem 12.2	300
Processing Data from an Array	242		
Analysis of the Flowchart in Figure 11.2	244		

Flexible Ways of Processing Arrays 305

FOR/NEXT with Negative Steps 307

String Arrays 307

Summary 308

BASIC Statements 308

Terms 309

Review Questions 309

Exercises 309

13 Nested Loops and Two-Dimensional Arrays 311

Nested Loops 311

Analysis of the Algorithms in Figure 13.2 311

Problem 13.1 314

Two-Dimensional Arrays 318

Building Two-Dimensional Arrays 320

Problem 13.2 324

Summary 336

Terms 336

Review Questions 336

Exercises 337

14 Algorithm Development: File Processing 339

Examples of File Processing in Manual Data Processing 339

File Structures 342

System Flowcharts 342

Creating a File 344

The Control (Key) Field 344

Definition of Sequence 346

Sequence Check to Previous Highest Control Field 347

Sequence Check Involving Each Successive Pair of Control Fields 351

Updating a File 353

Analysis of the Flowchart in Figure 14.14 357

Optional Approach for Updating 363

Merging Files 364

Analysis of the Flowchart in Figure 14.18 365

Control Break (Report Writing) 370

The Summary Report 373

The Detail Report 377

Summary 379

Terms 380

Review Questions 380

Exercises 380

File Creation 380

Sequence Checking (to previous highest control field) 381

Updating 381

Merging 381

Control Break (Report Writing) 381

General 382

15 Computer Programming: File Processing 383

Diskette Organization 383

INPUT Statement 385

Things to Know about INPUT

Statements 387

Using INPUT Statements 388

Prompting with INPUT Statements 389

Creating a File 389

Analysis of the Program in Figure 15.3 391

The Control (Key) Field 396

Sequence Check 397

Analysis of the Program in Figure 15.5 398

Updating a File 401

Analysis of the Program in Figure 15.8 408

Merging Files 416

Analysis of the Program in Figure 15.11 416

Control Break (Report Writing) 425

The Summary Report 425

The Detail Report 432

Summary 434

BASIC Statements 435

Terms 435

Review Questions 435

Exercises	435
File Creation	435
Sequence Checking (to previous highest key)	436
Updating	437
Merging	437
Control Break (Report Writing)	437
General	438

16 Formatting Printed Output 439

Vertical Spacing (The PRINT Statement)	440
---	------------

Horizontal Spacing (The TAB Statement)	441
---	------------

Formatting Output (The USING Statement)	444
Assigning Print Images to Variables	447
Printing Commas	449
Printing Dollar Signs	449
Printing Negative Number Notations	451
Printing String Data with USING	451

Positioning Numeric Data in Output	452
---	------------

Positioning String Data in Output	454
--	------------

Summary of Image Characters	454
------------------------------------	------------

Problem 16.1	455
Analysis of Problem 16.1	455
Solutions to Problem 16.1	455

Problem 16.2	457
Analysis of Problem 16.2	457
Solution to Problem 16.2	458

Summary	460
----------------	------------

BASIC Statements	460
-------------------------	------------

Terms	461
--------------	------------

Review Questions	461
-------------------------	------------

Exercises	461
------------------	------------

17 Debugging 463

Approach to Debugging	463
------------------------------	------------

The Emotions of Debugging	464
----------------------------------	------------

Types of Errors	464
Syntax Errors	464
Logic Errors	465

Trace Procedures	466
Walkthrough	466
Computer Trace	466
Programming Trace	472

Summary	474
----------------	------------

BASIC Statements	475
-------------------------	------------

Terms	475
--------------	------------

Review Questions	475
-------------------------	------------

Exercises	475
------------------	------------

18 Number and Coding Systems 479

Number Systems	479
Positional Value	480
Symbol Value	481
Base 10 (Decimal)	482
Base 2 (Binary)	483
Base 16 (Hexadecimal)	484
Converting a Decimal to a Binary Equivalent	484
Converting a Decimal to a Hexadecimal Equivalent	486
Converting a Binary to a Hexadecimal Equivalent	488
Converting a Hexadecimal to a Binary Equivalent	489

Coding Systems	492
Hollerith Code (Figure 18.5)	493
EBCDIC (Figure 18.5)	493
ASCII-8 (Figure 18.5)	496
ASCII-7 (Figure 18.5)	497
Code Ranges	498

Memory	498
Address	499
Parity Bit	499
Words	500

Magnetic Storage	501
-------------------------	------------

Summary	501
----------------	------------

Terms	501
--------------	------------

Review Questions	502
-------------------------	------------

Exercises	502
------------------	------------

Index	503
--------------	------------

Problem Solving

1

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
 3. Decision
 4. Output
- } Considered the *process* in a three-step
} procedure of Input-Process-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	decision	problem solving
bug	execute	process
business programmer	input	scientific programmer
data processing	output	testing
debug	problem	

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.