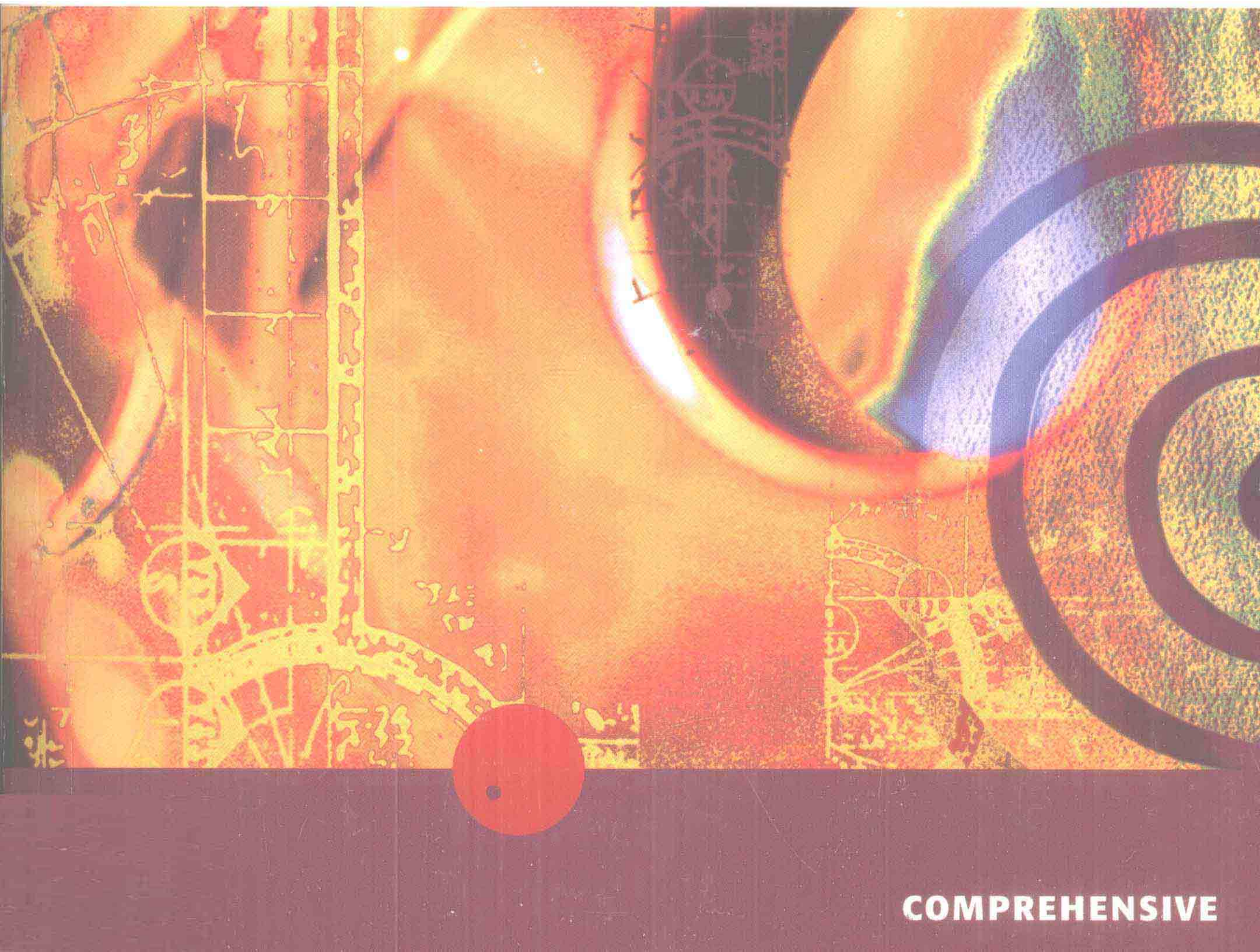
COURSE TECHNOLOGY
THOMSON LEARNING

Programming Logic and Design

SECOND EDITION

Joyce Farrell



COMPREHENSIVE

Programming Logic and Design, Comprehensive, Second Edition

Joyce Farrell
University of Wisconsin
Stevens Point

COURSE TECHNOLOGY
THOMSON LEARNING

Australia • Canada • Mexico • Singapore • Spain • United Kingdom • United States

COURSE TECHNOLOGY
THOMSON LEARNING™

Programming Logic and Design, Comprehensive, Second Edition

By Joyce Farrell

Product Manager:
Tricia Boyle

Managing Editor:
Jennifer Locke

Senior Editor:
Jennifer Muroff

Development Editor:
Lisa Ruffolo

Editorial Assistant:
Janet Aras

Production Editor:
Karen Jacot

Cover Designer:
Betsy Young

Compositor:
GEX Publishing Services

Manufacturing Coordinator:
Alexander Schall

ISBN 0-619-06315-7

Preface

Programming Logic and Design, Comprehensive, Second Edition, provides the beginning programmer with a guide to developing structured program logic. This textbook assumes that students have no programming language experience. The writing is nontechnical and emphasizes good programming practices. The examples are business examples; they do not assume mathematical background beyond high school business math. Additionally, the examples illustrate one or two major points; they do not contain so many features that students become lost following irrelevant and extraneous details.

The examples in *Programming Logic and Design, Comprehensive, Second Edition,* have been created to provide students with a sound background in logic no matter what programming languages they might eventually use to write programs. This book can be used in a stand-alone logic course that students take as a prerequisite to a programming course, or as a companion book to an introductory programming text using any programming language.

Organization and Coverage

Programming Logic and Design, Comprehensive, Second Edition, introduces students to programming concepts, enforcing good style and logical thinking. General programming concepts are introduced in Chapter 1. Chapter 2 discusses the key concepts of structure including what structure is, how to recognize it, and, most importantly, the advantages to writing structured programs. Chapter 3 extends the information on structured programming to the area of modules. By Chapter 4 students can write complete, structured business programs. Chapters 5 and 6 explore the intricacies of decision making and looping. Students learn to develop sophisticated programs that use control breaks and arrays in Chapters 7 and 8.

In Chapter 9 students use arrays in more sophisticated ways by exploring sorting techniques. Chapter 10 focuses on the special issues involved in writing interactive programs that allow users to make menu selections from both single- and multiple-level menus. Chapter 11 covers the intricacies of sequential file merging, matching, and updating. In Chapter 12, students learn valuable modularization techniques, including the differences between local and global variables and how to pass variables to and from modules. Chapter 12 also provides clear explanations of the sometimes confusing terminology associated with object-oriented programming. In Chapter 13, students learn the vocabulary associated with event-driven programming and how to incorporate GUI objects into programs. Chapter 14 brings together all the concepts that students have learned so far by addressing issues of good program design—including reducing coupling and increasing cohesion. Chapter 15 provides an introduction to UML, a powerful graphic tool for designing object-oriented systems.

Programming Logic and Design, Comprehensive, Second Edition, combines text explanation with flowcharts and pseudocode examples to provide students with alternative means of expressing structured logic. Numerous detailed, full-program exercises at the end of each section illustrate the concepts explained within the section and reinforce students' understanding and retention of the material presented.

Programming Logic and Design, Comprehensive, Second Edition, distinguishes itself from other programming logic language books in the following ways:

- It is written and designed to be non-language specific. The logic used in this book can be applied to any programming language.
- The examples are everyday business examples; no special knowledge of mathematics, accounting, or other disciplines is assumed.
- The concept of structure is covered earlier than in many other texts. Students are exposed to structure naturally, so they will automatically create properly designed programs.
- Text explanation is interspersed with both flowcharts and pseudocode so students can become comfortable with both logic development tools and understand their interrelationship.
- Complex programs are built through the use of complete business examples. Students see how an application is built from start to finish instead of studying only segments of programs. Students learn the difference between local and global variables, and how to pass their values to and from modules.
- Object-oriented terminology is thoroughly explained. This feature is absent from many other programming logic books.
- Event-driven GUI programs are presented. Students enjoy working with the graphical objects. Few texts explore the logic behind them.
- Students gain an appreciation for good program design, and learn to recognize poor design.
- Students learn about UML, a powerful object-oriented design tool.

FEATURES

New to this edition of the text is the use of camel casing for naming variables and methods, and the addition of parentheses to method names. These improvements expose students to variable and method names that closely resemble those they will be most likely to use in introductory programming classes in C++, Java, Visual Basic or Pascal. To improve students' comprehension of the decision-making process, endif has been added to the pseudocode in all decision examples. Additional end-of-chapter exercises are included so students have more opportunities to practice concepts as they learn them.

Programming Logic and Design, Comprehensive, Second Edition, is a superior textbook because it also includes the following features:

- **Objectives** Each chapter begins with a list of objectives so the student knows the topics that will be presented in the chapter. In addition to providing a quick reference to topics covered, this feature provides a useful study aid.
- **Tips** These notes provide additional information—for example, an alternative method of performing a procedure, another term for a concept, background information on a technique, or a common error to watch out for.
- **Summaries** Following each section is a summary that recaps the programming concepts and techniques covered in the section. This feature provides a concise means for the student to recap and check understanding of each chapter's main points.
- **Exercises** Each chapter section concludes with meaningful programming exercises that provide students with additional practice of the skills and concepts they learned in the lesson. These exercises increase in difficulty and are designed to allow students to explore logical programming concepts.

TEACHING TOOLS

The following supplemental materials are available when this book is used in a classroom setting. All of the teaching tools available with this book are provided to the instructor on a single CD-ROM.

Electronic Instructor's Manual. The Instructor's Manual that accompanies this textbook includes:

- Additional instructional material to assist in class preparation, including suggestions for lecture topics.
- Solutions to all end-of-chapter materials.

ExamView®. This textbook is accompanied by ExamView, a powerful testing software package that allows instructors to create and administer printed, computer (LAN-based), and Internet exams. ExamView includes hundreds of questions that correspond to the topics covered in this text, enabling students to generate detailed study guides that include page references for further review. The computer-based and Internet testing components allow students to take exams at their computers, and also save the instructor time by grading each exam automatically.

PowerPoint Presentations. This book comes with Microsoft PowerPoint slides for each chapter. These are included as a teaching aid for classroom presentation, to make available to students on the network for chapter review, or to be printed for classroom distribution. Instructors can add their own slides for additional topics they introduce to the class.

Solution Files. Solutions to end-of chapter exercises are provided on the Teaching Tools CD-ROM and may also be found on the Course Technology Web site at **www.course.com**. The solutions are password protected.

Distance Learning. Course Technology is proud to present online courses in WebCT and Blackboard, as well as at MyCourse.com, Course Technology's own course enhancement tool, to provide the most complete and dynamic learning experience possible. When you add online content to one of your courses, you're adding a lot: self tests, links, glossaries, and, most of all, a gateway to the 21st century's most important information resource. We hope you will make the most of your course, both online and offline. For more information on how to bring distance learning to your course, contact your local Course Technology sales representative.

ACKNOWLEDGMENTS

I would like to thank all of the people who helped to make this book a reality, especially Lisa Ruffolo, Development Editor, whose hard work, attention to detail, and perceptive appreciation for the methodology required to teach this subject matter have made this a quality textbook. Thanks also to Kristen Duerr, Vice President and Publisher; Tricia Boyle, Product Manager; Jennifer Locke, Managing Editor; Jennifer Muroff, Senior Editor; Margarita Donovan, Senior Product Manager; Karen Jacot, Production Editor; Janet Aras, Editorial Assistant; and Beverly Jackson, Quality Assurance Tester. I am grateful to be able to work with so many fine people who are dedicated to producing quality instructional materials.

I am grateful to the many reviewers who provided helpful and insightful comments during the development of this revision, including R. Scott Cost, University of Maryland at Catonsville; Bob Husson, Craven Community College; Cathleen Kennedy, College of San Mateo; Marilyn D. Moore, Morehead State University; George Novotny, Ferris State University; Steve Prettyman, Chattahoochee Technical Institute; and Pamela Silvers, Asheville-Buncombe Technical Community College.

Thanks, too, to my husband, Geoff, who acts as friend and advisor in the book-writing process. This book, as was its previous edition, is dedicated to him and to my daughters, Andrea and Audrey.

Joyce Farrell

Programming Logic and Design, Comprehensive, Second Edition, is a superior textbook because it also includes the following features:

- **Objectives** Each chapter begins with a list of objectives so the student knows the topics that will be presented in the chapter. In addition to providing a quick reference to topics covered, this feature provides a useful study aid.
- **Tips** These notes provide additional information—for example, an alternative method of performing a procedure, another term for a concept, background information on a technique, or a common error to watch out for.
- **Summaries** Following each section is a summary that recaps the programming concepts and techniques covered in the section. This feature provides a concise means for the student to recap and check understanding of each chapter's main points.
- **Exercises** Each chapter section concludes with meaningful programming exercises that provide students with additional practice of the skills and concepts they learned in the lesson. These exercises increase in difficulty and are designed to allow students to explore logical programming concepts.

TEACHING TOOLS

The following supplemental materials are available when this book is used in a classroom setting. All of the teaching tools available with this book are provided to the instructor on a single CD-ROM.

Electronic Instructor's Manual. The Instructor's Manual that accompanies this textbook includes:

- Additional instructional material to assist in class preparation, including suggestions for lecture topics.
- Solutions to all end-of-chapter materials.

ExamView®. This textbook is accompanied by ExamView, a powerful testing software package that allows instructors to create and administer printed, computer (LAN-based), and Internet exams. ExamView includes hundreds of questions that correspond to the topics covered in this text, enabling students to generate detailed study guides that include page references for further review. The computer-based and Internet testing components allow students to take exams at their computers, and also save the instructor time by grading each exam automatically.

PowerPoint Presentations. This book comes with Microsoft PowerPoint slides for each chapter. These are included as a teaching aid for classroom presentation, to make available to students on the network for chapter review, or to be printed for classroom distribution. Instructors can add their own slides for additional topics they introduce to the class.

BRIEF

Contents

PREFACE xi

CHAPTER ONE
An Overview of Computers and Logic 1

CHAPTER TWO
Understanding Structure 29

CHAPTER THREE
Modules, Hierarchy Charts, and Documentation 61

CHAPTER FOUR
Writing a Complete Program 91

CHAPTER FIVE
Making Decisions 111

CHAPTER SIX
Looping 149

CHAPTER SEVEN
Control Breaks 177

CHAPTER EIGHT
Arrays 209

CHAPTER NINE
Advanced Array Manipulation 247

CHAPTER TEN
Using Menus and Validating Input 279

CHAPTER ELEVEN
Sequential File Merging, Matching, and Updating 311

CHAPTER TWELVE
Advanced Modularization Techniques and
Object-Oriented Programming 343

CHAPTER THIRTEEN
Programming Graphical User Interfaces 371

CHAPTER FOURTEEN
Program Design 391

CHAPTER FIFTEEN
System Modeling With UML 411

APPENDIX A
A Difficult Structuring Problem A-1

APPENDIX B
Using a Large Decision Table B-1

INDEX I-1

TABLE OF
Contents

PREFACE	**xi**
CHAPTER ONE	
An Overview of Computers and Logic	**1**
Understanding Computer Components and Operations	1
Understanding the Programming Process	5
Understand the Problem	5
Plan the Logic	6
Code the Program	7
Translate the Program into Machine Language	7
Test the Program	8
Put the Program into Production	9
Understanding the Data Hierarchy	9
Using Flowchart Symbols and Pseudocode Statements	11
Using and Naming Variables	15
Ending a Program by Using Sentinel Values	17
Using the Connector	19
Assigning Values to Variables	20
Understanding Data Types	22
Chapter Summary	23
Exercises	24
CHAPTER TWO	
Understanding Structure	**29**
Understanding Unstructured Spaghetti Code	29
Understanding the Three Basic Structures	31
Using the Priming Read	38
Understanding the Reasons for Structure	42
Recognizing Structure	44
Two Special Structures—Case and Do Until	49
The Case Structure	49
The Do Until Loop	52
Chapter Summary	54
Exercises	55
CHAPTER THREE	
Modules, Hierarchy Charts, and Documentation	**61**
Modules, Subroutines, Procedures, Functions, or Methods	61
Modularization Provides Abstraction	62
Modularization Allows Multiple Programmers to Work on a Problem	62
Modularization Allows You to Reuse Your Work	63
Modularization Makes It Easier to Identify Structures	63
Modularizing a Program	66
Modules Calling Other Modules	69

Declaring Variables 71
Creating Hierarchy Charts 73
Understanding Documentation 75
Output Documentation 75
Input Documentation 81
Completing the Documentation 85
Chapter Summary 86
Exercises 87

CHAPTER FOUR
Writing a Complete Program 91
Understanding the Mainline Logical Flow Through a Program 91
Housekeeping Tasks 94
- Declaring Variables 94
- Opening Files 99
- Printing Headings 100
- Reading the First Input Record 100

Writing the Main Loop 104
Performing End-Of-Job Tasks 107
Chapter Summary 107
Exercises 108

CHAPTER FIVE
Making Decisions 111
Evaluating Boolean Expressions to Make Comparisons 111
Using the Logical Comparison Operators 114
Understanding AND Logic 118
Writing AND Decisions for Efficiency 122
Combining Decisions in an AND Situation 124
Avoiding Common Errors in an AND Situation 125
Understanding OR Logic 127
Avoiding Common Errors in an OR Situation 129
Writing OR Decisions for Efficiency 130
Combining Decisions in an OR Situation 131
Using Selections within Ranges 132
Common Errors Using Range Checks 133
Using Decision Tables 135
Chapter Summary 142
Exercises 143

CHAPTER SIX
Looping 149
Understanding the Advantages of Looping 149
Using a While Loop with a Loop Control Variable 150
Using a Counter to Control Looping 152
Looping with a Variable Sentinel Value 156
Looping by Decrementing 158
Avoiding Common Loop Mistakes 159
Using the For Loop 161
Using the Do Until Loop 163
Recognizing the Characteristics Shared by All Loops 165
Nesting Loops 166

Using a Loop to Accumulate Totals 168
Chapter Summary 172
Exercises 173

CHAPTER SEVEN
Control Breaks 177
Understanding Control Break Logic 177
Performing Single-Level Control Breaks 178
Using Control Data Within the Control Break Module 184
Performing Control Breaks with Totals 189
Performing Multiple-Level Control Breaks 194
Performing Page Breaks 202
Chapter Summary 205
Exercises 206

CHAPTER EIGHT
Arrays 209
Understanding Arrays 209
How Arrays Occupy Computer Memory 210
Manipulating an Array to Replace Using Nested Decisions 211
Array Declaration and Initialization 221
Run-Time and Compile-Time Arrays 223
Loading an Array from a File 226
Searching for an Exact Match in an Array 228
Using Parallel Arrays 230
Remaining within Array Bounds 233
Improving Search Efficiency using an Early Exit 235
Searching an Array for a Range Match 237
Chapter Summary 240
Exercises 241

CHAPTER NINE
Advanced Array Manipulation 247
Understanding the Need for Sorting Records 247
Understanding How to Swap Two Values 248
Using a Bubble Sort 250
Refining the Bubble Sort by Using a Variable for the Array Size 258
Refining the Bubble Sort by Reducing Unnecessary Comparisons 262
Refining the Bubble Sort by Eliminating Unnecessary Passes Through the List 263
Using an Insertion Sort 265
Using a Selection Sort 266
Using Indexed Files 268
Using Linked Lists 269
Using Multidimensional Arrays 271
Chapter Summary 275
Exercises 276

CHAPTER TEN
Using Menus and Validating Input 279
Using Interactive Programs 279
Using a Single-Level Menu 281

Coding Modules as Black Boxes 285
Making Improvements to a Menu Program 289
Using the Case Structure to Manage a Menu 293
Using Multilevel Menus 296
Validating Input 301
Understanding Types of Data Validation 303
Validating a Data Type 304
Validating a Data Range 304
Validating Reasonableness and Consistency of Data 305
Validating Presence of Data 305
Chapter Summary 306
Exercises 307

CHAPTER ELEVEN
Sequential File Merging, Matching and Updating 311
Understanding Sequential Data Files and the Need for Merging Files 311
Creating the Mainline and Housekeeping() Logic for a Merge Program 313
Creating the mainLoop() and finishUp() Modules for a Merge Program 316
Modifying the housekeeping() Module to Check for eof 321
Master and Transaction File Processing 323
Matching Files to Update Fields in Master File Records 324
Allowing Multiple Transactions for a Single Master File Record 328
Updating Records in Sequential Files 329
Chapter Summary 338
Exercises 339

CHAPTER TWELVE
Advanced Modularization Techniques and Object-Oriented Programming 343
Understanding the Principles of Modularization and Abstraction in Procedural Programs 343
Passing Variables to Modules 348
Returning a Value from a Module 352
Using an IPO Chart 354
Understanding the Advantages of Encapsulation 354
An Overview of Object-Oriented Programming 356
Defining Classes 358
Instantiating and Using Objects 360
Understanding Inheritance 361
Understanding Polymorphism 363
The Advantages of Object-Oriented Programming 366
Chapter Summary 366
Exercises 367

CHAPTER THIRTEEN
Programming Graphical User Interfaces 371
Understanding Event-Driven Programming 371
User-Initiated Actions and GUI Components 373
Designing Graphical User Interfaces 375
The Interface Should Be Natural and Predictable 375
The Interface Should Be Attractive, Easy to Read, and Nondistracting 375
To Some Extent, It's Helpful If The User Can Customize Your Applications 376
The Program Should Be Forgiving 376

The GUI Is Only a Means to an End 376
Modifying the Attributes of GUI Components 377
The Steps to Developing an Event-Driven Application 377
Understanding the Problem 378
Creating Storyboards 378
Defining the Objects in an Object Dictionary 379
Defining the Connections Between the User Screens 380
Planning the Logic 381
Object-Oriented Error Handling: Throwing Exceptions 382
Chapter Summary 387
Exercises 388

CHAPTER FOURTEEN
Program Design 391
Understanding the Need for Good Program Design 391
Storing Program Components in Separate Files 391
Selecting Variables and Module Names 394
Designing Module Statements 396
Avoid Confusing Line Breaks 396
Use Temporary Variables to Clarify Long Statements 398
Use Constants where Appropriate 399
Organizing Modules 400
Reducing Coupling 400
Increasing Cohesion 403
Functional Cohesion 403
Sequential Cohesion 404
Communicational Cohesion 405
Temporal, Procedural, Logical, and Coincidental Cohesion 406
Maintaining Good Programming Habits 407
Chapter Summary 408
Exercises 408

CHAPTER FIFTEEN
System Modeling With UML 411
Understanding the Need for System Modeling 411
What is UML? 412
Using Use Case Diagrams 414
Using Class and Object Diagrams 420
Using Sequence and Collaboration Diagrams 425
Using Statechart Diagrams 427
Using Activity Diagrams 428
Using Component and Deployment Diagrams 430
Deciding which UML Diagrams to Use 432
Chapter Summary 432
Exercises 434

APPENDIX A A-1
A Difficult Structuring Problem

APPENDIX B B-1
Using a Large Decision Table

INDEX I-1

CHAPTER

1 An Overview of Computers and Logic

After studying Chapter 1, you should be able to:

- ♦ Understand computer components and operations
- ♦ Describe the steps involved in the programming process
- ♦ Describe the data hierarchy
- ♦ Understand how to use flowchart symbols and pseudocode statements
- ♦ Use and name variables
- ♦ Use a sentinel, or dummy value, to end a program
- ♦ Use a connector symbol
- ♦ Assign values to variables
- ♦ Recognize the proper format of assignment statements
- ♦ Describe data types

Understanding Computer Components and Operations

The two major components of any computer system are its hardware and its software. **Hardware** is the equipment, or the devices, associated with a computer. For a computer to be useful, however, it needs more than equipment; a computer needs to be given instructions. The instructions that tell the computer what to do are called **software**, or programs, and are written by programmers. This book focuses on the process of writing these instructions.

Together, computer hardware and software accomplish four major operations:

1. Input
2. Processing
3. Output
4. Storage

Hardware devices that perform input include keyboards and mice. Through these devices, **data**, or facts, enter the computer system. Processing data items may involve organizing them, checking them for accuracy, or performing mathematical operations on them. The piece of hardware that performs these sorts of tasks is the **Central Processing Unit,** or **CPU**. After data have been processed, the resulting information is sent to a printer, monitor, or some other output device. Often, you also want to store the output information on hardware, such as magnetic disks or tapes. Computer software consists of all the instructions that control how and when the data are input, how they are processed, and the form in which they are output or stored.

Computer hardware by itself is useless without a programmer's instructions or software, just as your stereo equipment doesn't do much until you provide music on a CD or tape. You can enter instructions into a computer system through any of the hardware devices you use for data: for example, a keyboard or disk drive.

You write computer instructions in a computer **programming language** such as Visual Basic, Pascal, COBOL, RPG, C#, C++, Java, or Fortran. Just as some humans speak English and others speak Japanese, programmers also write programs in different languages. Some programmers work exclusively in one language, while others know several and use the one that seems most appropriate for the task at hand.

No matter which programming language a computer programmer uses, the language has rules governing its word usage and punctuation. These rules are called the language's **syntax**. If you ask, "How the get to store do I?" in English, most people can figure out what you probably mean even though you have not used proper English syntax. However, computers are not nearly as smart as most humans; with a computer you might as well have asked, "Xpu mxv ot dodnm cadf B?" Unless the syntax is perfect, the computer cannot interpret the programming language instruction at all.

Every computer operates on circuitry that consists of millions of on-off switches. Each programming language uses a piece of software to translate the specific programming language into the computer's on-off circuitry language, or **machine language**. The language translation software is called a **compiler** or **interpreter**, and it tells you if you have used a programming language incorrectly. Therefore, syntax errors are relatively easy to locate and correct. If you write a computer program using a language such as C++, but spell one of its words incorrectly or reverse the proper order of two words, the translator lets you know it found a mistake as soon as you try to run the program.

Tip

Although there are differences in how compilers and interpreters work, their basic function is the same—to translate your programming statements into code the computer can use.

For a program to work properly, you must give the instructions to the computer in a specific sequence, you must not leave any instructions out, and you must not add extraneous instructions. By doing this, you are developing the **logic** of the computer program. Suppose you instruct someone to make a cake as follows:

```
Stir
Add two eggs
Add a gallon of gasoline
Bake at 350 degrees for 45 minutes
Add three cups of flour
```

Even though you have used the English language syntax correctly, the instructions are out of sequence, some instructions are missing, and some instructions belong to procedures other than baking a cake. If you follow these instructions, you are not going to end up with an edible cake, and you may end up with a disaster. Logical errors are much more difficult to locate than syntax errors; it is easier for you to determine whether *eggs* is spelled incorrectly in a recipe than it is for you to tell if there are too many eggs or they are added too soon.

Just as baking directions can be given correctly in French, German, or Spanish, the same logic of a program can be expressed in any number of programming languages. This book is almost exclusively concerned with the logic development process. Because it is not concerned with any specific language, this book could have been written in Japanese, C++, or Java. The logic is the same in any language. For convenience, the book uses English!

Once instructions have been input to the computer and translated into machine language, a program can be **run** or **executed**. You can write a program that takes a number (an input step), doubles it (processing), and tells you the answer (output) in a programming language such as Pascal or C++, but if you were to write it in English, it would look like this:

```
Get inputNumber
Compute calculatedAnswer as inputNumber times 2
Print calculatedAnswer
```

The instruction to `Get inputNumber` is an example of an input operation. When the computer interprets this instruction, it knows to look to an input device to obtain a number. Computers often have several input devices, perhaps a keyboard, a mouse, a CD drive, and two or more disk drives. When you learn a specific programming language, you learn how to tell the computer which of those input devices to access for input. For now, however, it doesn't really matter which hardware device is used as long as the computer knows to look for a number. The logic of the input operation—that the computer must obtain a number for input, and that the computer must obtain it before multiplying it by two—remains the same regardless of any specific input hardware device.

Tip

Many computer professionals categorize disk drives and CD drives as storage devices rather than input devices. Such devices actually can be used for input, storage, and output.