Eighth Edition

C++ PROGRAMMING

Program Design Including Data Structures

D.S. Malik

CENGAGE Learning

ISBN-13: 978-1-337-11756-2
ISBN-10: 1-337-11756-0
90000
9 781337 117562

D.S. Malik

C++ PROGRAMMING

Program Design Including Data Structures

Eighth Edition

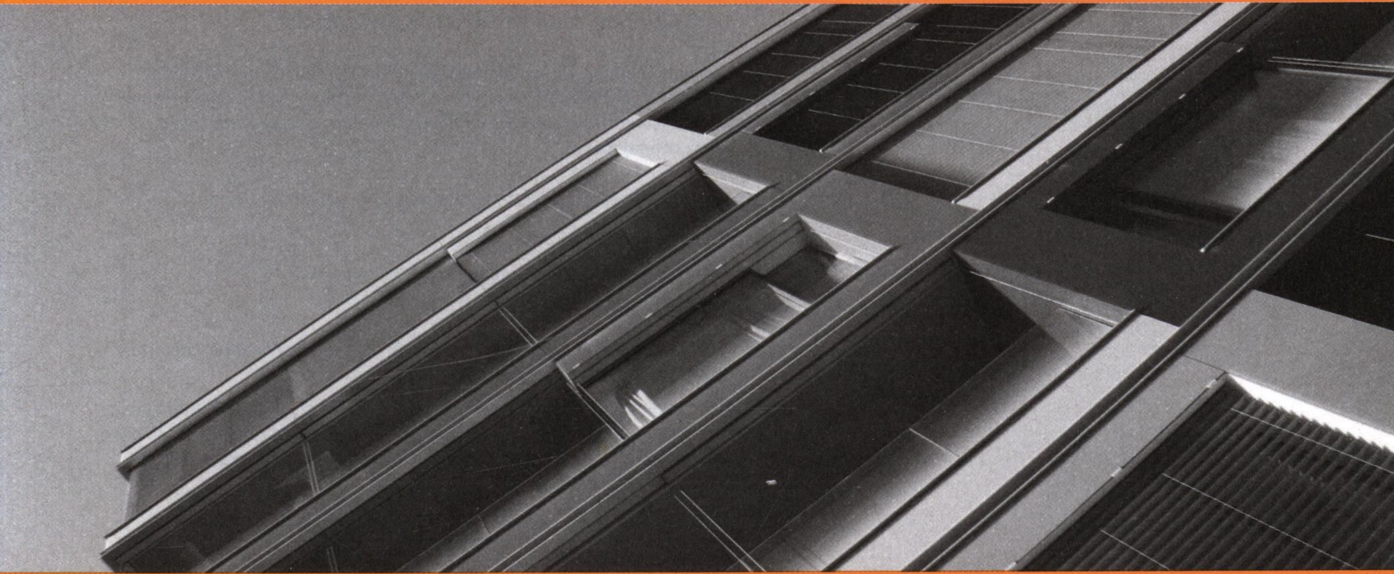
CENGAGE Learning

C++ Programming:

Program Design Including Data Structures

Eighth Edition

D.S. Malik



CENGAGE Learning

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

CENGAGE
Learning

C++ Programming: Program Design Including Data Structures, Eighth Edition

D.S. Malik

Senior Product Director: Kathleen McMahon

Product Team Leader: Kristin McNary

Associate Product Manager: Kate Mason

Associate Content Development Manager: Alyssa Pratt

Production Director: Patty Stephan

Senior Content Project Manager: Jennifer Feltri-George

Manufacturing Planner: Julio Esperas

Art Director/Cover Design: Diana Graham

Production Service/Composition: SPi Global

Cover Photo: Cebas/Shutterstock.com

For product information and technology assistance, contact us at
Cengage Learning Customer & Sales Support, 1-800-354-9706

For permission to use material from this text or product,
submit all requests online at **www.cengage.com/permissions**.
Further permissions questions can be emailed to
permissionrequest@cengage.com.

Library of Congress Control Number: 2017931947

ISBN: 978-1-337-11756-2

Cengage Learning
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at **www.cengage.com**.

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

To learn more about Cengage Learning Solutions, visit **www.cengage.com.**

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com.**

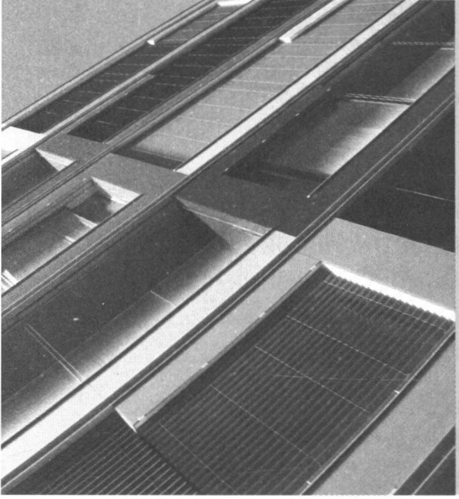
Any fictional data related to persons or companies or URLs used throughout this book is intended for instructional purposes only. At the time this book was printed, any such data was fictional and not belonging to any real persons or companies.

The programs in this book are for instructional purposes only. They have been tested with care, but are not guaranteed for any particular intent beyond educational purposes. The author and the publisher do not offer any warranties or representations, nor do they accept any liabilities with respect to the programs.

Printed in the United States of America
Print Number: 01 Print Year: 2017

TO

My Parents



© HunThomas/Shutterstock.com

Preface

WELCOME TO THE EIGHTH EDITION OF *C++ Programming: Program Design Including Data Structures*. Designed for a first Computer Science (CS1 and CS2) C++ course, this text provides a breath of fresh air to you and your students. The CS1 and CS2 course serves as the cornerstone of the Computer Science curriculum. My primary goal is to motivate and excite all introductory programming students, regardless of their level. Motivation breeds excitement for learning. Motivation and excitement are critical factors that lead to the success of the programming student. This text is a culmination and development of my classroom notes throughout more than fifty semesters of teaching successful programming to Computer Science students.

Warning: This text can be expected to create a serious reduction in the demand for programming help during your office hours. Other side effects include significantly diminished student dependency on others while learning to program.

C++ Programming: Program Design Including Data Structures started as a collection of brief examples, exercises, and lengthy programming examples to supplement the books that were in use at our university. It soon turned into a collection large enough to develop into a text. *The approach taken in this book is, in fact, driven by the students' demand for clarity and readability*. The material was written and rewritten until the students felt comfortable with it. Most of the examples in this book resulted from student interaction in the classroom.

As with any profession, practice is essential. Cooking students practice their recipes. Budding violinists practice their scales. New programmers must practice solving problems and writing code. This is not a C++ cookbook. We do not simply list the C++ syntax followed by an example; we dissect the "why?" behind all the concepts. The crucial question of "why?" is answered for every topic when first introduced. This technique offers a bridge to learning C++ Students must understand the "why?" in order to be motivated to learn.

Traditionally, a C++ programming neophyte needed a working knowledge of another programming language. This book assumes no prior programming experience. However, some adequate mathematics background, such as college algebra, is required.

Changes in the Eighth Edition

The eighth edition contains more than 300 new and updated exercises, requiring new solutions, and more than 20 new programming exercises.

This edition also introduces C++14 digit separator (Chapter 3), C++11 class inline functions (Chapter 10), updated C++11 class data members initialization during declaration (Chapter 10), and C++11 random generators (Chapter 13). The C-string functions such as `strcpy`, `strcmp`, and `strcat` have been deprecated, and might give warning messages when used in a program. Furthermore, the functions `strncpy` and `strncmp` might not be implemented in all versions of C++ Therefore, in Chapter 13, we have modified the Programming Example `newString` to reflect these changes by including functions to copy a character array.

Approach

The programming language C++, which evolved from C, is no longer considered an industry-only language. Numerous colleges and universities use C++ for their first programming language course. C++ is a combination of structured programming and object-oriented programming, and this book addresses both types.

This book is intended for a two-semester course, CS1 and CS2, in Computer Science. The first 10 or 11 chapters can be covered in the first course and the remaining in the second course.

In July 1998, ANSI/ISO Standard C++ was officially approved. This book focuses on ANSI/ ISO Standard C++. Even though the syntax of Standard C++ and ANSI/ISO Standard C++ is very similar, Chapter 7 discusses some of the features of ANSI/ISO Standard C++ that are not available in Standard C++.

Chapter 1 briefly reviews the history of computers and programming languages. The reader can quickly skim through this chapter and become familiar with some of the hardware components and the software parts of the computer. This chapter contains a section on processing a C++ program. This chapter also describes structured and object-oriented programming.

Chapter 2 discusses the basic elements of C++ After completing this chapter, students become familiar with the basics of C++ and are ready to write programs that are complicated enough to do some computations. Input/output is fundamental to any programming language. It is introduced early, in Chapter 3, and is covered in detail.

Chapters 4 and 5 introduce control structures to alter the sequential flow of execution. Chapter 6 studies user-defined functions. It is recommended that readers with no prior programming background spend extra time on Chapter 6. Several examples are provided to help readers understand the concepts of parameter passing and the scope of an identifier.

Chapter 7 discusses the user-defined simple data type (enumeration type), the `namespace` mechanism of ANSI/ISO Standard C++ and the **`string`** type. The earlier versions of C did not include the enumeration type. Enumeration types have very limited use; their main purpose is to make the program readable. This book is organized such that readers can skip the section on enumeration types during the first reading without experiencing any discontinuity, and then later go through this section.

Chapter 8 discusses arrays in detail. This chapter also discusses range-based `for` loops, a feature of C++11 Standard, and explains how to use them to process the elements of an array. Limitations of ranged-based `for` loops on arrays passed as parameters to functions are also discussed. Chapter 8 also discusses a sequential search algorithm and a selection sort algorithm. Chapter 9 introduces records (`struct`s). The introduction of `struct`s in this book is similar to C `struct`s. This chapter is optional; it is not a prerequisite for any of the remaining chapters.

Chapter 10 begins the study of object-oriented programming (OOP) and introduces classes. The first half of this chapter shows how classes are defined and used in a program. The second half of the chapter introduces abstract data types (ADTs). The inline functions of a classes are introduced in this chapter. Also, the section "In-Class Initialization of Data Members and the Default Constructor" has been updated. Furthermore, this chapter shows how classes in C++ are a natural way to implement ADTs. Chapter 11 continues with the fundamentals of object-oriented design (OOD) and OOP and discusses inheritance and composition. It explains how classes in C++ provide a natural mechanism for OOD and how C++ supports OOP. Chapter 11 also discusses how to find the objects in a given problem.

Chapter 12 studies pointers in detail. After introducing pointers and how to use them in a program, this chapter highlights the peculiarities of classes with pointer data members and how to avoid them. Moreover, this chapter discusses how to create and work with dynamic two-dimensional arrays, and also explains why ranged-based `for` loops cannot be used on dynamic arrays. Chapter 12 also discusses abstract classes and a type of polymorphism accomplished via virtual functions.

Chapter 13 continues the study of OOD and OOP. In particular, it studies polymorphism in C++. The chapter specifically discusses two types of polymorphism—overloading and templates. Moreover, C++11 random number generators are introduced in this chapter.

Chapter 14 discusses exception handling in detail. Chapter 15 introduces and discusses recursion. Moreover, this is a stand-alone chapter, so it can be studied anytime after Chapter 9.

Chapters 16 and 17 are devoted to the study of data structures. Discussed in detail are linked lists in Chapter 16 and stacks and queues in Chapter 17. The programming code developed in these chapters is generic. These chapters effectively use the fundamentals of OOD.

Chapter 18 discusses various searching and sorting algorithms. In addition to showing how these algorithms work, it also provides relevant analysis and results concerning the performance of the algorithms. The algorithm analysis allows the user to decide which algorithm to use in a particular application. This chapter also includes several sorting algorithms. The instructor can decide which algorithms to cover.

Chapter 19 provides an introduction to binary trees. Various traversal algorithms, as well as the basic properties of binary trees, are discussed and illustrated. Special binary trees, called binary search trees, are introduced. Searching, as well as item insertion and deletion from a binary search tree, are described and illustrated. Chapter 19 also discusses nonrecursive binary tree traversal algorithms. Furthermore, to enhance the flexibility of traversal algorithms, it shows how to construct and pass functions as parameters to other functions. This chapter also discusses AVL (height balanced) trees in detail. Because of text length considerations, discussion on AVL trees is provided as a separate section and is available on the website accompanying this book.

Graph algorithms are discussed in Chapter 20. After introducing the basic graph theory terminology, the representation of graphs in computer memory is discussed. This chapter also discusses graph traversal algorithms, the shortest path algorithm, and the minimal spanning tree algorithm. Topological sort is also discussed in this chapter and is available on the website accompanying this book.

C++ is equipped with a powerful library—the Standard Template Library (STL)—of data structures and algorithms that can be used effectively in a wide variety of applications. Chapter 21 describes the STL in detail. After introducing the three basic components of the STL, it shows how sequence containers are used in a program. Special containers, such as stacks and queues, are also discussed. The latter half of this chapter shows how various STL algorithms can be used in a program. This chapter is fairly long; depending on the availability of time, the instructor can at least cover the sequence containers, iterators, the classes `stack` and `queue`, and certain algorithms.

Appendix A lists the reserved words in C++. Appendix B shows the precedence and associativity of the C++ operators. Appendix C lists the ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code) character sets. Appendix D lists the C++ operators that can be overloaded.

Appendix E, provided online, has three objectives. First, we discuss how to convert a number from decimal to binary and binary to decimal. We then discuss binary and random access files in detail. Finally, we describe the naming conventions of the header files in both ANSI/ISO Standard C++ and Standard C++. Appendix F discusses some of the most widely used library routines, and includes the names of the standard C++ header files. The programs in Appendix G show how to print the memory size for the built-in data types on your system. Appendix H gives selected references for further study. Appendix I provides the answers to odd-numbered exercises in the book.

How to Use the Book

This book can be used in various ways. Figure 1 shows the dependency of the chapters.

In Figure 1, dotted lines mean that the preceding chapter is used in one of the sections of the chapter and is not necessarily a prerequisite for the next chapter. For example, Chapter 8 covers arrays in detail. In Chapters 9 and 10, we show the relationship between arrays and `struct`s and arrays and classes, respectively. However, if Chapter 10 is studied before Chapter 8, then the section dealing with arrays in Chapter 10 can be skipped without any discontinuation. This particular section can be studied after studying Chapter 8.

It is recommended that the first six chapters be covered sequentially. After covering the first six chapters, if the reader is interested in learning OOD and OOP early, then Chapter 10 can be studied right after Chapter 6. Chapter 7 can be studied anytime after Chapter 6.

After studying the first six chapters in sequence, some of the approaches are:

1. Study chapters in the sequence: 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 , 19, 20, 21.
2. Study chapters in the sequence: 8, 10, 12, 13, 11, 15, 16, 17, 14, 18, 19, 20, 21.
3. Study chapters in the sequence: 10, 8, 12, 13, 11, 15, 16, 17, 14, 18, 19, 20, 21.

As the chapter dependency diagram shows, Chapters 17 and 18 can be covered in any sequence. However, typically, Chapters 17 and 18 are studied in sequence. Ideally, one should study Chapters 16, 17, 18, and 19 in sequence. Chapters 20 and 21 can be studied in any sequence.

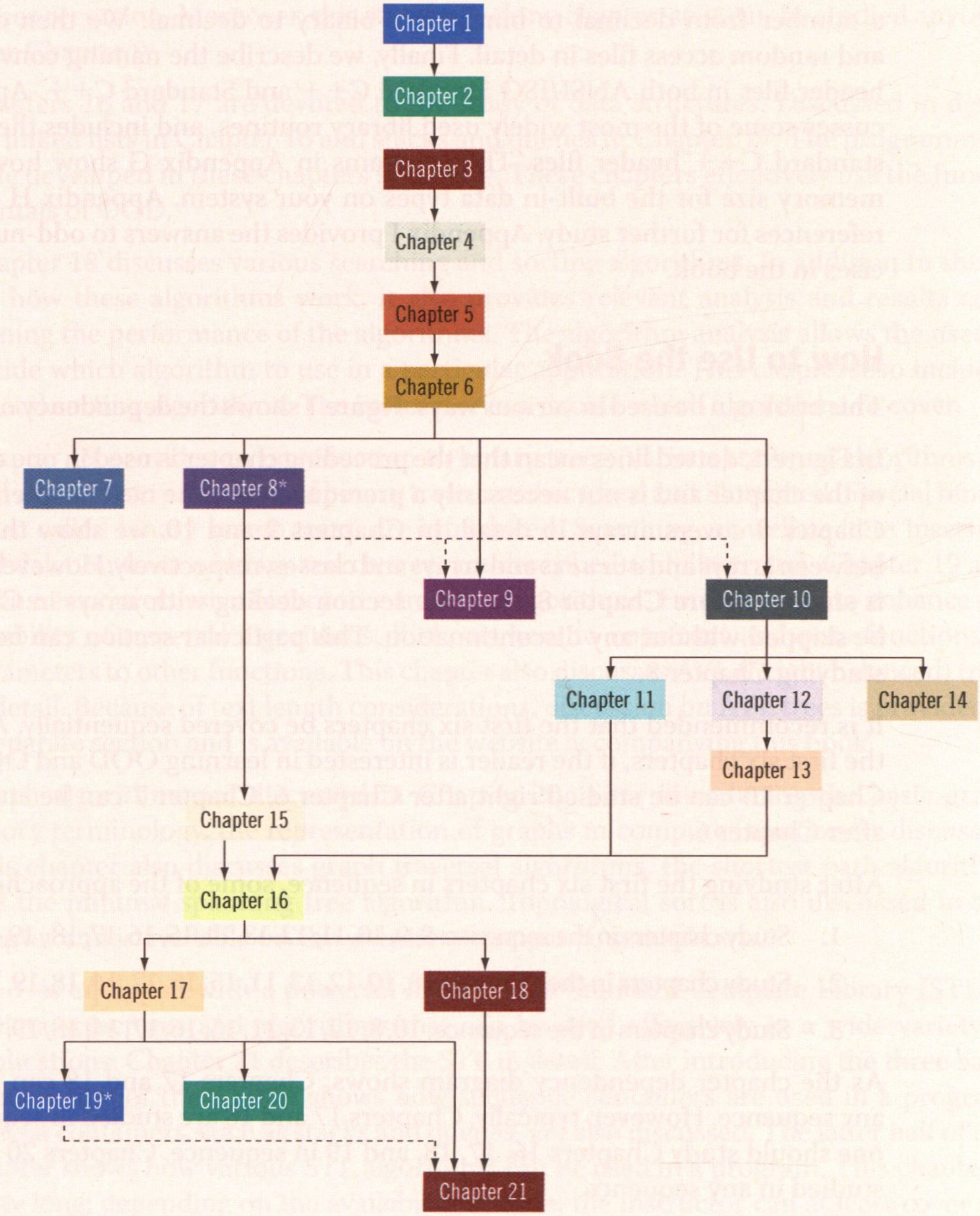
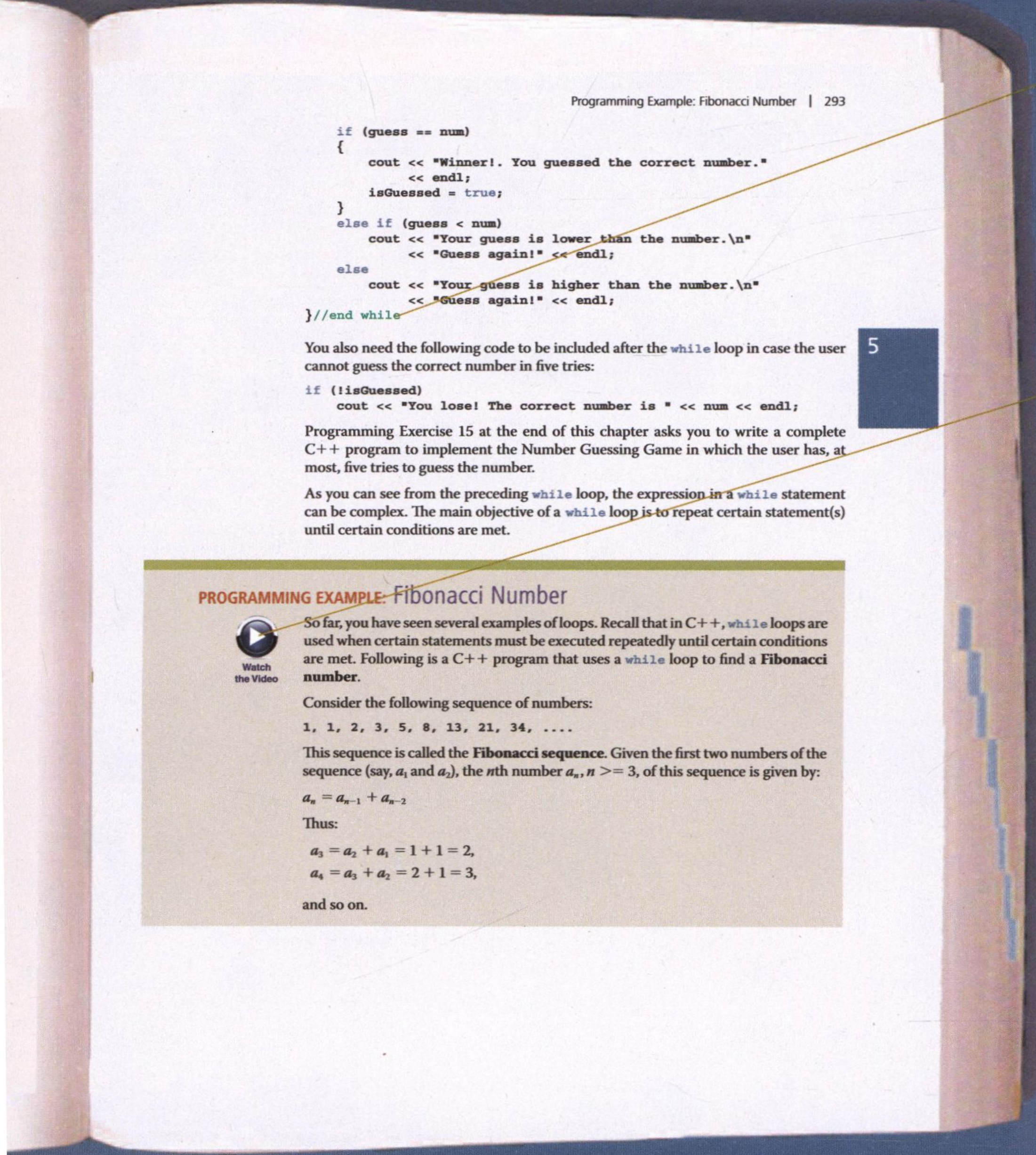


FIGURE 1 Chapter dependency diagram

Features of the Book



```
    if (guess == num)
    {
        cout << "Winner!. You guessed the correct number."
              << endl;
        isGuessed = true;
    }
    else if (guess < num)
        cout << "Your guess is lower than the number.\n"
              << "Guess again!" << endl;
    else
        cout << "Your guess is higher than the number.\n"
              << "Guess again!" << endl;
}//end while
```

You also need the following code to be included after the while loop in case the user cannot guess the correct number in five tries:

```
if (!isGuessed)
    cout << "You lose! The correct number is " << num << endl;
```

Programming Exercise 15 at the end of this chapter asks you to write a complete C++ program to implement the Number Guessing Game in which the user has, at most, five tries to guess the number.

As you can see from the preceding while loop, the expression in a while statement can be complex. The main objective of a while loop is to repeat certain statement(s) until certain conditions are met.

PROGRAMMING EXAMPLE: Fibonacci Number

Watch the Video

So far, you have seen several examples of loops. Recall that in C++, while loops are used when certain statements must be executed repeatedly until certain conditions are met. Following is a C++ program that uses a while loop to find a **Fibonacci number**.

Consider the following sequence of numbers:

```
1, 1, 2, 3, 5, 8, 13, 21, 34, ....
```

This sequence is called the **Fibonacci sequence**. Given the first two numbers of the sequence (say, a_1 and a_2), the *n*th number a_n, $n >= 3$, of this sequence is given by:

$$a_n = a_{n-1} + a_{n-2}$$

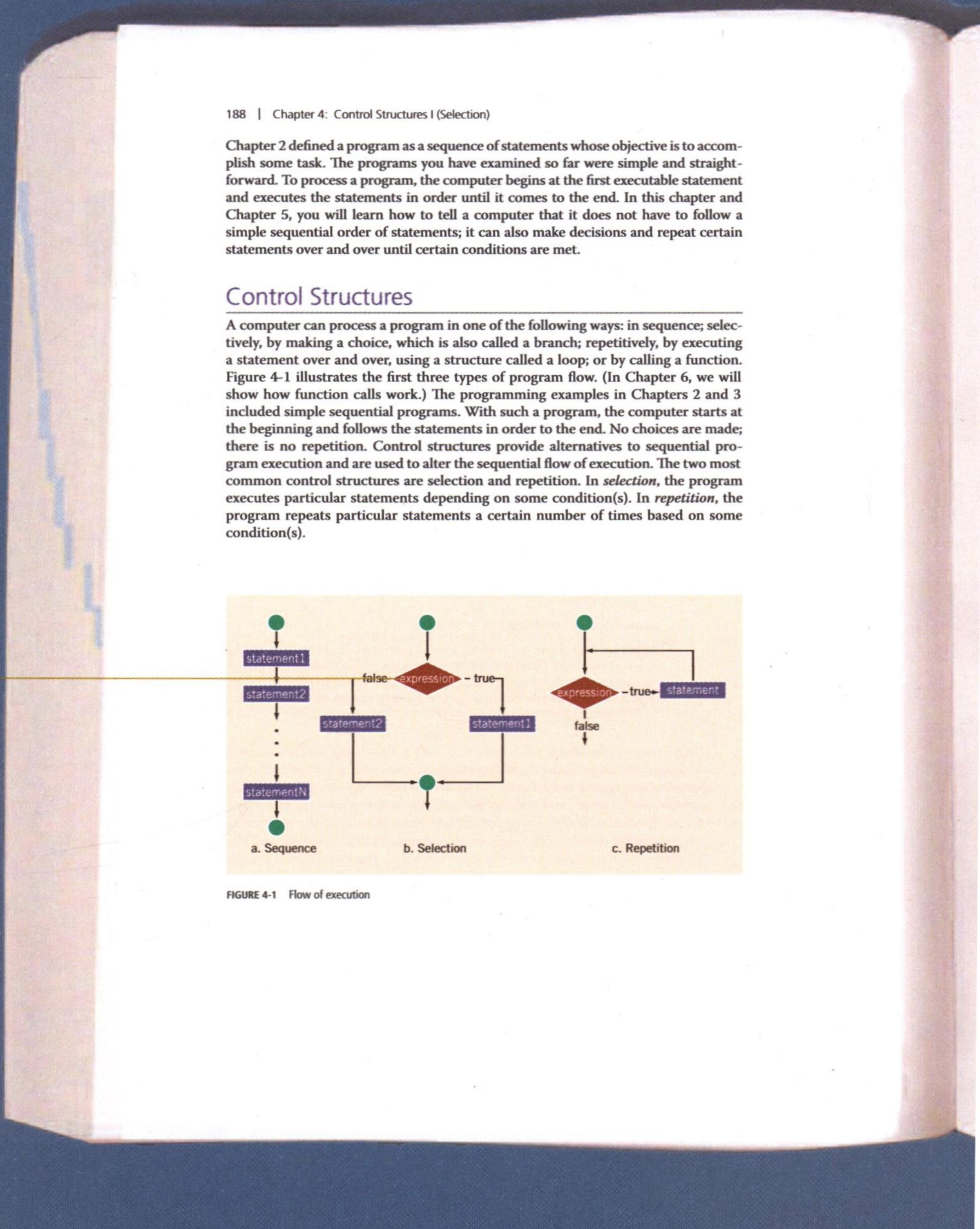
Thus:

$$a_3 = a_2 + a_1 = 1 + 1 = 2,$$
$$a_4 = a_3 + a_2 = 2 + 1 = 3,$$

and so on.

Four-color interior design shows accurate C++ code and related comments.

One video is available for each chapter on CengageBrain .com. Each video is designed to explain how a program works.



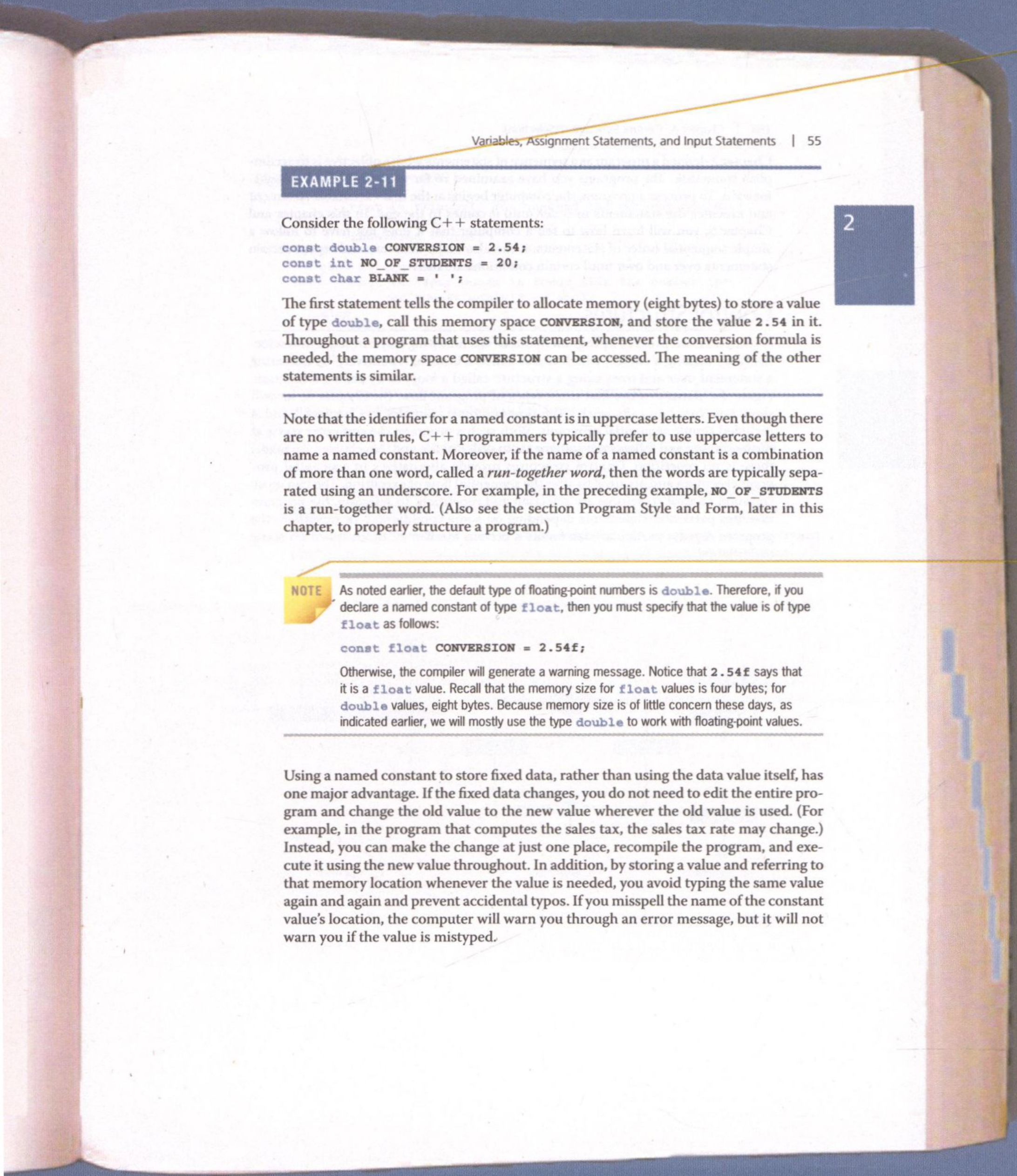
Chapter 2 defined a program as a sequence of statements whose objective is to accomplish some task. The programs you have examined so far were simple and straightforward. To process a program, the computer begins at the first executable statement and executes the statements in order until it comes to the end. In this chapter and Chapter 5, you will learn how to tell a computer that it does not have to follow a simple sequential order of statements; it can also make decisions and repeat certain statements over and over until certain conditions are met.

Control Structures

A computer can process a program in one of the following ways: in sequence; selectively, by making a choice, which is also called a branch; repetitively, by executing a statement over and over, using a structure called a loop; or by calling a function. Figure 4-1 illustrates the first three types of program flow. (In Chapter 6, we will show how function calls work.) The programming examples in Chapters 2 and 3 included simple sequential programs. With such a program, the computer starts at the beginning and follows the statements in order to the end. No choices are made; there is no repetition. Control structures provide alternatives to sequential program execution and are used to alter the sequential flow of execution. The two most common control structures are selection and repetition. In ***selection***, the program executes particular statements depending on some condition(s). In ***repetition***, the program repeats particular statements a certain number of times based on some condition(s).

FIGURE 4-1 Flow of execution

More than 300 visual diagrams, both extensive and exhaustive, illustrate difficult concepts.



EXAMPLE 2-11

2

Consider the following C++ statements:

```
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
```

The first statement tells the compiler to allocate memory (eight bytes) to store a value of type `double`, call this memory space `CONVERSION`, and store the value `2.54` in it. Throughout a program that uses this statement, whenever the conversion formula is needed, the memory space `CONVERSION` can be accessed. The meaning of the other statements is similar.

Note that the identifier for a named constant is in uppercase letters. Even though there are no written rules, C++ programmers typically prefer to use uppercase letters to name a named constant. Moreover, if the name of a named constant is a combination of more than one word, called a *run-together word*, then the words are typically separated using an underscore. For example, in the preceding example, `NO_OF_STUDENTS` is a run-together word. (Also see the section Program Style and Form, later in this chapter, to properly structure a program.)

NOTE As noted earlier, the default type of floating-point numbers is `double`. Therefore, if you declare a named constant of type `float`, then you must specify that the value is of type `float` as follows:

```
const float CONVERSION = 2.54f;
```

Otherwise, the compiler will generate a warning message. Notice that `2.54f` says that it is a `float` value. Recall that the memory size for `float` values is four bytes; for `double` values, eight bytes. Because memory size is of little concern these days, as indicated earlier, we will mostly use the type `double` to work with floating-point values.

Using a named constant to store fixed data, rather than using the data value itself, has one major advantage. If the fixed data changes, you do not need to edit the entire program and change the old value to the new value wherever the old value is used. (For example, in the program that computes the sales tax, the sales tax rate may change.) Instead, you can make the change at just one place, recompile the program, and execute it using the new value throughout. In addition, by storing a value and referring to that memory location whenever the value is needed, you avoid typing the same value again and again and prevent accidental typos. If you misspell the name of the constant value's location, the computer will warn you through an error message, but it will not warn you if the value is mistyped.

Numbered Examples illustrate the key concepts with their relevant code. The programming code in these examples is followed by a Sample Run. An explanation then follows that describes what each line in the code does.

Notes highlight important facts about the concepts introduced in the chapter.