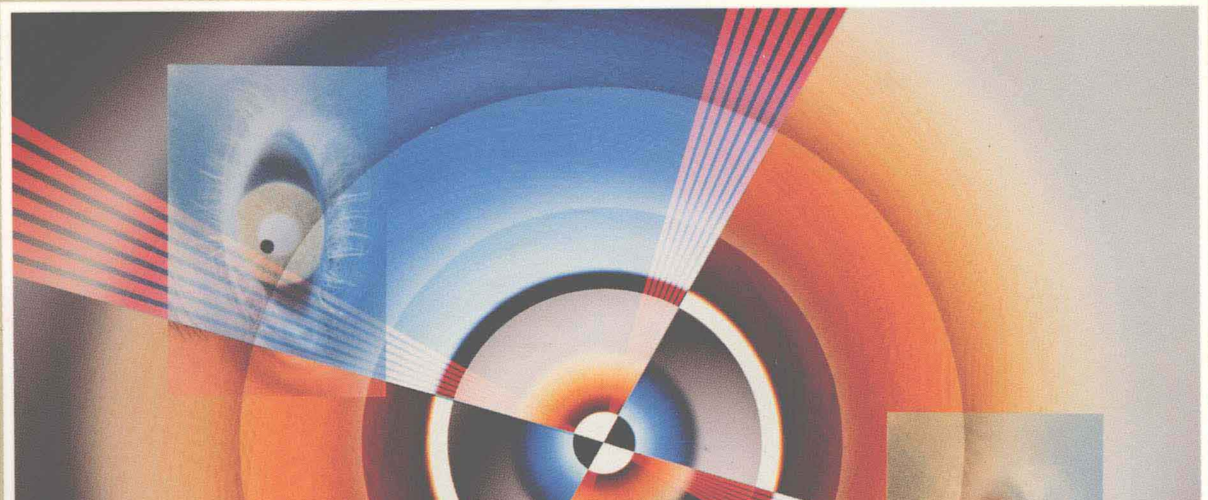
PROGRAMMING IN C++

An Applied Approach



HABIB T. KASHANI

Programming in C++
An Applied Approach

Habib T. Kashani
Langara College

TP312/K18

Prentice Hall

Upper Saddle River, New Jersey   *Columbus, Ohio*

Library of Congress Cataloging-in-Publication Data

Kashani, Habib T.
Programming in C++: an applied approach/Habib T. Kashani.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-228818-4
1. C++ (Computer program language) I. Title.
QA76.73.C153K37 1998
005.13'3--dc21 97-17979
CIP

Cover photo: Digital Art, Westlight
Editor: Charles E. Stewart, Jr.
Production Editor: Stephen C. Robb
Design Coordinator: Karrie M. Converse
Cover Designer: Rod Harris
Production Manager: Patricia A. Tonneman
Illustrations: Custom Editorial Productions, Inc.
Production Supervision: Custom Editorial Productions, Inc.
Marketing Manager: Debbie Yarnell

This book was set in Times Roman by Custom Editorial Productions, Inc., and was printed and bound by Quebecor Printing/Book Press. The cover was printed by Phoenix Color Corp.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN: 0-13-228818-4

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada, Inc., *Toronto*
Prentice-Hall Hispanoamericana, S. A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

To my wife, Nahid, and my children, Mathew, Suzanne, and Cyrus.

Preface

Programming in C++ is intended to meet the needs of both the student and the professional programmer. This book may be used in a second programming course, after an introductory course in computers and programming, for students with computer science majors and students with other majors. It may also be used as a supplement to an introductory computer science or software engineering text, if the topics take students into abstract data types and object-oriented programming. In addition, this text is recommended for students of computer information systems (CIS), business, technology, and continuing education programs. Finally, professional programmers who want to learn how to write programs in C++ and those who are looking for a resource reference will find this text useful.

PEDAGOGICAL APPROACH

This book's thorough treatment of C++ includes a comprehensive view of the features of the language as they apply to both structured programming and object-oriented programming. The book's focus is the fusion of concepts and applications; its purpose is to demonstrate how to apply theory to practical problems. To that end, concepts move from simple to complex so that each one is a foundation for the next level. Each concept is discussed in a clear and concise manner to make the learning process enjoyable, fast, and effective.

The discussion of each concept includes a variety of examples, which demonstrate how to apply the concepts and reinforce the principles of good programming. Each major topic is followed by a list of self-check questions and exercises, for which answers are supplied at the end of the book. Moreover, since the best way to learn how to write good programs is to see well-written ones, several sample programs, with internal documentation and necessary comment lines, are included at the end of each chapter. The sample programs are deliberately placed at the end to provide full discussion of the material without interruption, making the discussions more cohesive and understandable. Although the programs presented here have been tested using the Borland C++ compiler, they were designed to work without modification on almost any C++ compiler.

Finally, the style tips at the end of each chapter tell students how to write programs that are easy to read and maintain. They include schemes for using meaningful identifiers and specifying their case and symbol set (e.g., DaysOfWeek or days_of_week or Days_Of_Week) as well as methods for documenting the program, its interfaces, and classes. The intent behind this initiative is to develop and establish a consistent style in writing code. Good programming style also makes the programmer a more productive team member in a joint development project.

ORGANIZATION OF THE BOOK

Programming in C++: An Applied Approach is organized into six chapters, each of which covers a group of logically related topics. The chapters cover almost all facilities of the language in an orderly manner and explain the mechanical underpinnings of C++ supported by application examples for each topic. This innovative approach assures easy progression for the novice and offers a clear reference for the professional.

- Chapter 1 provides the fundamental programming concepts of C++. To ensure thorough understanding, readers should study the material in the sequence in which it is presented. The chapter also covers in detail the standard *I/O* facilities of the language.
- Chapter 2 focuses on more advanced topics, including language constructs, arrays and strings, pointers and references, and storage management.
- Chapter 3 discusses functions and describes in considerable detail how they interact with each other, the way in which they are invoked, and the values that are returned, as well as storage classes, scope rules, and lifetime of variables.
- Chapter 4 covers the features that help readers develop more advanced applications. Topics include enumerated data types, the type definition construct (*typedef*), different types of C++ functions, library functions, the preprocessor, and command line arguments.
- Chapter 5 addresses the structures and techniques for file processing. Structures and unions, bitfields, classes as unique C++ features, and an in-depth discussion of files are the topics of this chapter. Because there is some variability in readers' comprehension of these topics, the chapter is aimed at presenting them with the means to understand thoroughly, to design readily, and to implement easily the structures and files.
- Chapter 6 introduces the notion of an Abstract Data Type (ADT) and its practical use in object-oriented programming. The chapter gives students a clear understanding of the advantages of object-oriented programming before they invest time and effort to learn how to do it. To that end, it presents many ADTs of different levels of complexity. Finally, the chapter takes the reader through the design and implementation of each ADT in the following steps:

  Specifications
  Representation
  Implementation
  Application

Since the most important advance offered by C++ is its support for object-oriented programming, the level of detail in this chapter is considerably higher in order to address adequately the prominent features of the language, including:

- Classes
- Constructors and Destructors
- Inheritance
- The Keyword *friend*
- Virtual Functions and Dynamic Binding
- Polymorphism
- Pure Functions and Abstract Classes
- Static Members
- Operator Overloading
- Type Conversion
- Generic Classes Using Templates and Void Data Types
- Embedded Objects

ACKNOWLEDGMENTS

I would like to thank the following students at Langara College for their comments and suggestions: Balash Akbari, Kristian Erickson, Raymond Chan, Yan Teplitsky, and Graham TerMarsch.

In addition, I express sincere appreciation to the following colleagues who provided valuable input for the manuscript: Judy Boxler, Michael Kuttner, Pierre Massicotte, and Fred Parvaz.

I also thank executive editor Charles E. Stewart for his encouragement, support, and patience, and the other good people at Prentice Hall, Steve Robb, Kate Linsner, and JoEllen Gohr.

Finally, I gratefully acknowledge the contribution of these reviewers for their comments, criticism, and helpful suggestions: Michael Bush, Carl C. Hommer, Jr., Usha Jindal, and Philip Regalbuto.

Habib Kashani
Langara College

Brief Contents

Contents

1 Fundamental Concepts

1.1 PREVIEW

In this chapter you will learn:

- A Historical Overview of Programming
- The Origin of the C++ Language
- Program Structure
- The Elements of C++: Functions, Identifiers, Keywords, Literals, and Operators
- Expressions
- Data Types
- Variable Declaration and Scopes
- Standard Input-Output Operations

1.2 A HISTORICAL OVERVIEW OF PROGRAMMING

Over the past forty years programming languages have gone through an evolutionary change. The journey started with assembly language. The first assemblers made programming a relative pleasure by providing mnemonic names for operation codes (such as *mul* for multiplication operations and *add* for addition operations) and by allowing programmers to refer to memory locations by symbolic names.

Advances in computer technology, including those in both hardware and software on the one hand and experience in programming and software development on the other, led to the appearance of high-level languages in the 1960s. The focus of programming has shifted from compact code to structured programming using procedures and functions. Many new programming languages have emerged in a short period of time to support this concept of using data types, control statements, functions, procedures, and modules. Programs became structured and procedural abstraction became pivotal to software design and the programming paradigm. For the past twenty-five years, structured programming has worked very well for environments in which programs tended to be relatively stable and the code did not change significantly. Almost all of the business and scientific applications of this period have operated under these criteria.

Recent achievements in microprocessor architecture and data communication technology have created unprecedented computing power and opportunities for the software industry. In the last decade alone, applications for computers have increased dramatically and literally thousands of software packages have been developed for every purpose imaginable. New areas for software applicability arise each week, and applications are required to be more functional. The continuing demand for ambitious applications with many capabilities have led to programs that are voluminous and complex.

The volume and complexity of applications demand a new approach to software design and engineering. The software for today's applications must be

- Reliable
- Reusable
- Easy to develop
- Easy to maintain

Code reliability plays a critical role in software design. Reliable code is correct and robust: Correct code does exactly what the specifications state and robust software handles exceptional cases in a reasonable way without sudden program termination.

Ability to reuse code increases the programmer's productivity and facilitates development, allowing him or her to use as much as possible of the existing functionality in writing new applications. Ease of maintenance implies that an application ought to be flexible, extendable, and readable. Maintenance has always played an important role in software development. This role in today's applications is even more critical in light of continuing changes in hardware and software requirements. A maintainable application allows easy modification, refinement, and improvement.

1.3 THE ORIGIN OF THE C++ LANGUAGE

C++ is a general-purpose programming language based on the C programming language. In 1972, the language C was designed for programming under the UNIX operating system. The language was named C because it was based on an earlier version called B. Both B and C represent different versions designed after the earlier systems programming language BCPL (Basic Compound Programming Language). The primary difference was that B, the first letter of BCPL, was an essentially typeless language, while C, the second letter of BCPL, had an extensive collection of standard types. In 1973 UNIX itself was extended, and more than 90 percent of it was rewritten from assemby language in C. The current versions of the C language are mostly based on the ANSI Standard C. For more information, obtain a copy of a publication titled "American National Standard Information Interchange—Programming Language C" (1990) from the American National Standards Institutes in New York.

The C programming language has attracted considerable attention internationally because of its popularity in the software industry. The reasons for this popularity are its power, flexibility, efficiency, compactness, and portability. However, the need for greater modularity within programs and support for the development of large and complex systems with maximum efficiency led to the evolutionary development of C++. Developed by Dr. Bjarne Stroustrup at the Computer Science Research Center at AT&T Bell Laboratories in Murray Hill, New Jersey, and this language became available in 1985. The cryptic name C++ implies that it is an enhanced version or superset of its predecessor C.

C++ is not a completely new language. It can be thought of more as an evolutionary advancement of C. Both languages share the fundamental concepts for using statements, data types, operators, function definition, and separate compilation. The primary aim in extending C++ has been to enhance it as a suitable language for data abstraction and object-oriented programming. Data abstraction, in contrast to procedural abstraction, is a new approach to programming and software development. To this end, C++ has provided a stable platform for implementing these concepts and developing high-quality tools for complex environments. (The topics of data abstraction and object-oriented programming will be discussed later in Chapter 6.)

Dr. Stroustrup implemented the language as a C++-to-C translator called CFRONT. CFRONT translated the C++ code into C code so that it could be compiled and linked in the traditional way. Today, there are CFRONT ports as well as full compilers available for the C++ language. In contrast to CFRONT ports, full compilers such as Borland C++ generate object code directly rather than going through a conversion process.

1.4 PROGRAM STRUCTURE

A typical C++ program consists of two parts: the **global part** and **the main function part.** The global part contains necessary declarations, definitions, and files needed by the entire program. The main function, referred to as *main( ),* is the most important part as the program execution starts with this function. We begin by examining a simple program that displays a one-line greeting on the terminal screen.

The first step is to create a file that contains the program. The file can be named something like *hello.c, hello.cpp, hello.cp,* or *hello.cxx,* depending on the conventions required by the compiler used. For example, the Turbo C++ compiler requires *filename.cpp.*

```
// File name: hello.cpp
// This program displays a greeting to the user.

#include <iostream.h>   // for input and output operations

void main ()            // main program part
{
cout << "Hello, Reader!";
}
```

A terminal session with this program looks like this:

```
Hello, Reader!
```

The first two lines comprise comment lines in C++. Comments are helpful remarks that appear in the program listing but have no effect on the way the program runs. Their purpose is to make programs more readable. The comment after the // symbols extends to the end of the line. C++ also provides another commenting style, one in which the line starts with /* (slash and asterisk) and ends with */ (asterisk and slash).

The third and fifth lines are blank for clarity. Line 4 includes the **Stream I/O header file** which, among several other things, contains the definitions needed to allow the program to read input and write output. A typical C++ program normally starts with this unique feature, which is known as a **preprocessor statement.** It is an instruction to the compiler to retrieve the necessary code from the Stream I/O header file into the source code on the line requested. The **cout** (standard output stream) object of this file is used for output. Header files normally have a *.h* extension. In some implementations, the extension may be *.hpp* or *.hxx.* We will explore these entities in greater depth later in this chapter.

The sixth, seventh, and ninth lines include the usual *begin* and *end* of the **main** function of the program. Note that a C++ program consists of one or more functions as the basic building blocks. The function *main( )* is the major function that points to the place in the program where execution starts. Moreover, each function name is followed by a pair of parentheses to indicate that the name refers to a function and not another construct.

The body of the function—a group of statements that specify the actions to be performed—is enclosed between two **curly braces** as shown on the seventh and ninth lines. The purpose of the *cout* statement on line 8 is to print out the desired message. Notice that this statement is indented within the braces. Proper indentation of statements is a style matter. Although it makes no difference to the compiler for the translation of the program from human-oriented form (source code) to machine-readable format (object code), it helps us read the program more easily. Programming style will be discussed throughout this book. In addition, there is a style guide summary at the end of each chapter.

Note: C++ is case sensitive. Unless otherwise specified, all keywords of the language are to be written in lower-case letters. User-defined names can be either in lower- or upper-case letters.

We will now look at another short program to show how computations are performed in C++. The following program takes the length and width of a field and computes the number of meters of fence wire required to enclose the field.

```
//  File name: fence.cpp
//  This program computes how many meters of fence is
//  required for a rectangular field.

#include <iostream.h> // Allows the program to read input and
                      // write output.
void main()           // the main function of the program
{            // The opening brace indicates the beginning of the program.

//  Declaration part: shows the constants and variables used in
//  the program.

const int two = 2; // A constant declaration: two is of type int

int length, width, perimeter; // integer variables
```