COMPUTING CONCEPTS WITH C++ ESSENTIALS

CONTAINS ALL ESSENTIAL C++ NUTRIENTS
PREPARED WITH THE FINEST ANSI STRINGS AND VECTORS
OBJECT FARMS
OBJECT CHIP COOKIES
PACKED WITH PRACTICAL PROGRAMMING TIPS!

CAY HORSTMANN

Computing Concepts with C^{++} Essentials

Cay S. Horstmann
San Jose State University

John Wiley & Sons, Inc.
New York ◆ Chichester ◆ Brisbane ◆ Toronto ◆ Singapore ◆ Weinheim

ACQUISITIONS EDITOR	Regina Brooks
MARKETING MANAGER	Jay Kirsch
SENIOR PRODUCTION EDITOR	Tony VenGraitis
DESIGN DIRECTOR	Madelyn Lesure
MANUFACTURING MANAGER	Mark Cirillo
PHOTO EDITOR	Lisa Passmore
ILLUSTRATION COORDINATOR	Sandra Rigby

This book was set in Stempel Schneidler by Publication Services, Inc., and printed and bound by R. R. Donnelley/Crawfordsville. The cover was printed by Phoenix Color.

Recognizing the importance of preserving what has been written, it is a policy of John Wiley & Sons, Inc. to have books of enduring value published in the United States printed on acid-free paper, and we exert our best efforts to that end.

The paper in this book was manufactured by a mill whose forest managements programs include sustained yield harvesting of its timberlands. Sustained yield harvesting principles ensure that the number of trees cut each year does not exceed the amount of new growth.

Library of Congress Cataloging in Publication Data

Horstmann, Cay S 1959

Computing concepts with C++ essentials / Cay S Horstmann.

p cm

Includes bibliographical references

ISBN 0 471 13770 7 (alk paper)

1 C++ (Computer program language) 2 Computers I Title.

QA76 73 C153H669 1996 96 39307

005 13'3-dc21 CIP

Printed in the United States of America

10 9 8 7 6 5 4 3

Preface

This book gives a traditional introduction to computer science using modern tools. As computer scientists, we have the good fortune of being able to introduce students to an activity that is accessible, satisfying, and deep rather than broad: namely, the activity of *programming*. Like the majority of computer scientists, I believe that programming is the central theme of computer science. Thus, this course teaches students how to program.

While this book remains traditional in outlook, it uses modern techniques in three ways.

First, the programming language is a subset of C++. Although C++ is far from a perfect educational language, it makes pragmatic sense to use it. C++ is required for advanced computer science courses. Convenient and inexpensive programming environments are available on all major platforms. C++ is sufficiently expressive to teach programming concepts. This book avoids pointers through the use of modern features of the ANSI C++ standard—in particular, reference parameters, the stream library, the `string` class, and the `vector<T>` template. Pointers are used only for linked lists and polymorphism.

The second modern aspect is the early use of objects. Objects are introduced in two stages. From Chapter 2 on, students learn to *use* objects—in particular strings, streams, and graphical shapes. Students become comfortable with the concepts of creating objects and calling member functions. Then the book follows a traditional path, discussing branching and loops, functions, and procedures. Chapter 8 teaches how to *implement* classes and member functions.

The third modern aspect is the use of graphics. Students enjoy programming graphics. There are many programming examples in which the numbers and the visual information reinforce each other. This book uses a very simple graphics library that is available on a number of popular platforms. Unlike traditional graphics libraries, this library uses objects in a very straightforward and effective way.

The choice of programming language has a very visible impact on any book on programming. However, the purpose of the book is to teach computing concepts, not C++, which is just a tool toward that end. In 1997 there will be a Java version of this book that teaches the same concepts, in the same order, using *Java instead of* C++.

Sample Curricula

This book contains more material than could be covered in one semester, so you will need to make a choice of chapters to cover. The core material of the book is:

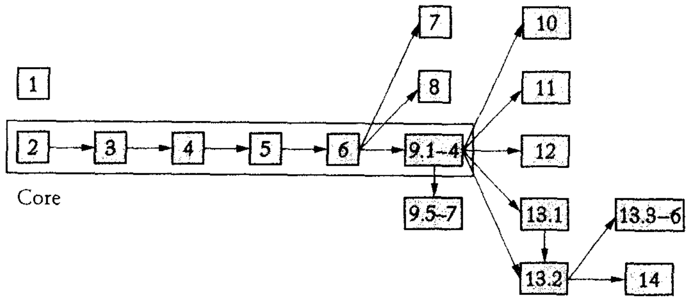
Chapter 2.	Fundamental Data Types
Chapter 3.	Objects
Chapter 4.	Decisions
Chapter 5.	Iteration
Chapter 6.	Functions
Sections 9.1–9.4	Arrays

The following chapters are optional:

Chapter 1.	Introduction
Chapter 7.	Testing and Debugging
Chapter 8.	Classes
Chapter 10.	Files
Chapter 11.	Modules
Chapter 12.	Algorithms
Chapter 13.	Data Structures
Chapter 14.	Inheritance

The following graph shows the dependencies between the chapters.

Dependencies Between Sections



The material in Chapters 12 though 14 is meant as an introduction to these topics, not a definitive treatment.

A good choice for a semester-long course is to teach Chapters 2 though 11. This will comfortably lead students into a second-semester course on data structures. If you are not concerned about separate compilation, you can omit Chapter 11 and cover Chapter 12 instead.

If the second semester uses C instead of C++, you may want to omit Chapter 11 and instead cover Appendix 3 to get students up to speed in C.

If the second semester covers object-oriented design in detail, you can omit Chapter 8, Sections 9.5 though 9.7, and Chapters 13 and 14. Your students will then be using classes, but they will not implement them.

If you choose to cover inheritance but skip Chapter 13, you will need to cover Section 13.2 on pointers. That section is independent of the remainder of Chapter 13.

The book covers the following knowledge units from the ACM curriculum guidelines.

AL1: Basic Data Structures (6 of 13 hours)
AL2: Abstract Data Types (2 of 2 hours)
AL3: Recursion (2 of 3 hours)
AL6: Sorting and Searching (2 of 6 hours)
PL3: Representation of Data Types (2 of 2 hours)
PL4: Sequence Control (2 of 4 hours)
PL5: Data Control, Sharing and Type Checking (2 of 4 hours)
PL6: Run-Time Storage Management (2 of 4 hours)
PR: Introduction to a Programming Language (12 of 12 hours)
SE1: Fundamental Problem-Solving Concepts (16 of 16 hours)
SP1: Historical and Social Context of Computing (3 of 3 hours)

The Use of C++

It is impossible to teach all of C++ to beginning programmers in one semester. This book purposefully omits a number of useful but technically complex C++ topics, such as copy construction and destruction, run-time type identification, function pointers, parameterized types, exception handling, name spaces, and operator overloading.

This book does not teach programming "close to the metal". For example, strings are treated as fundamental types. Characters are simply strings of length 1. This approach may offend those who feel that students should understand the C implementation of strings, but it has been a great hit with students, who no longer have to deal with character array overruns and dangling character pointers. There is exactly one place in the book where I missed the `char` type. (Challenge: Find it.)

Since the focus of the book is on computing concepts, I try not to dwell on syntactical wrinkles of C++. Consider the `#include` directive. In my opinion, the distinction between `<...>` and `"..."` delimiters is just the kind of clutter that serves no useful purpose in an introductory course. The directive `#include "iostream.h"` will find the system header just fine (provided there is no file named iostream.h in the current directory). If you think it is not a great burden to learn about the distinction between system headers and private headers, there is an Advanced Topic note in the book to back you up.

C++ has two styles of comments: the C-style /**/ and the C++-style //. The // style is great if you want to dash off a one-line comment, but it is a true pain for comments extending over multiple lines. I don't want to use two comment styles in the book. Furthermore, I would like to encourage students to write comments *as long as they need to be* rather than cramming them into the remainer of the current line. Therefore, I chose to use the /* */ comments. Naturally, everyone has a strong personal preference in this regard. It is an exceedingly minor matter, and if your preference doesn't match mine, please feel free to use whatever style you like best.

Occasionally, a choice of programming style was prompted by anticipating widespread use of the ANSI standard C++ libraries. For example, iterators are pointerlike objects that describe locations in containers. For technical reasons, iterators do not support the -> operator. Therefore, I chose to use the more mundane `(*p).m` for pointers as well. Again, if you don't agree, feel free to use the other syntax.

About the Code Library

To use this book, you need a copy of the code library. You can obtain the library via the World Wide Web (http://www.horstmann.com/ccc). Simply uncompress the library and place all files into your working directory. Then include the header file ccc.h (which stands for Computing Concepts with C++) in your programs. There is no need to build a project or to link in a separate library.

This book uses the ANSI C++ Boolean type, string class, and vector template throughout. Since not all compilers currently support these features, the code library that accompanies this book contains an implementation that has been tested on most major C++ compilers. The header file ccc.h includes the compiler version or the book library version of the string and header file, as appropriate.

The ccc.h header defines a small number of the utility functions to make life simpler for the students. For example, a function `round (x)` does the same thing as `(int)(x + 0.5)`. A function `uppercase` returns a string with all lowercase characters turned into uppercase—a feature that is inexplicably missing in the standard library. Look at Appendix 2, Section A2.15 for a listing of these functions. There are also definitions for π and e.

The graphics library is purposefully kept simple. There are just four shapes: points, lines, circles, and text. To display a shape, make an object and send it to the graphics window: `cwin << circle`. Students can learn the entire library in an hour. The library has been ported to several major platforms.

There are two simple classes, `Time` and `Employee`, that are used for many programming examples. They add some realism and reinforce the concept that most real-life programs do not just manipulate numbers and strings. These two are automatically included with the ccc.h header.

How to Use This Book

The material in this book is divided into three parts: the essential, the useful, and the optional. To cover the essential material, simply skip over all side notes. It is perfectly reasonable to ignore all side notes completely during lectures and assign them for home reading.

Three of the side note sets are useful for the students, namely the Common Errors, Productivity Hints, and Quality Tips. Students quickly discover the Common Errors and read them on their own. You may need to encourage them to read the Quality Tips. The Productivity Hints may be challenging to some students, but those with some computer experience tend to find them very helpful.

The Random Facts and Advanced Topics are optional. The Random Facts provide historical and social information on computing, as required to fulfill the "historical and social context" requirements of the ACM curriculum guidelines. Most students will read the Random Facts on their own while pretending to follow the lecture. You will need to suggest those Advanced Topics that you think are important. By the way, not all of them are truly "advanced". Occasionally, alternative syntax (such as the // comment delimiter) is explained in an Advanced Topic.

Most examples are in the form of complete, ready-to-run programs. The programs are available electronically, and you can give them to your students.

Appendix 1 contains a style guide for use with this book. I have found it highly beneficial to require a consistent style for all assignments. I realize that my style may be different from yours. If you have strong feelings about a particular issue, or if this style guide conflicts with local customs, feel free to modify it. The style guide is available in electronic form for this purpose.

Appendix 2 contains a summary of all C++ constructs and library functions and classes used in this book.

To make it possible to smuggle this book into a curriculum that is otherwise based on C or low-level C++, Appendix 3 contains a crash course in C that shows how the higher-level C++ features map to C.

Acknowledgments

I would like to thank my acquisitions editor Steven Elliot for encouraging me to write this book, and my editor Regina Brooks at John Wiley & Sons for guiding the book through the development process. Her many constructive suggestions greatly improved this book, as did those of the reviewers: John Cordero, University of Southern California; Wanda Dann, Syracuse University; David G. Kay, University of California, Irvine; Ron McCarty, Pennsylvania State University; Don McKnight, Los Medanos College; Bruce Mielke, University of Wisconsin, Green Bay;

and Charles M. Williams, Georgia State University. Thanks to my student assistants, Michael Carney, Darrel Cherry, and Jiaoyang Zhou, with their help with the random facts and the code library, and to Lisa Passmore of John Wiley & Sons for her photo research. At Publication Services, Jerome Colburn did a splendid job with the copyediting; Jan Fisher adroitly handled the communications with John Wiley & Sons; Kathryn Wright kept the production process well coordinated; and the art was rendered expertly by Paul Edwards and his studio team. John Johnson of Teapot Graphics designed the stunning cover. My love and apologies go once again to my family, who had to suffer through yet another book project that never seemed to end.

Contents

Preface *v*

Chapter 1 Introduction 1

1.1 What Is a Computer? *2*
1.2 What Is Programming? *2*
1.3 The Anatomy of a Computer *3*
1.4 Translating Human-Readable Programs to Machine Code *12*
1.5 Programming Languages *14*
1.6 Programming Language Design and Evolution *15*
1.7 Becoming Familiar with Your Computer *18*
1.8 Compiling a Simple Program *22*
1.9 Errors *25*
1.10 The Compilation Process *27*
1.11 Algorithms *30*

Chapter 2 Fundamental Data Types 37

2.1 Number Types *38*
2.2 Input and Output *46*
2.3 Assignment *49*
2.4 Constants *55*
2.5 Arithmetic *59*
2.6 Strings *66*

Chapter 3 Objects 89

3.1 Constructing Objects 90
3.2 Using Objects 92
3.3 Real-Life Objects 97
3.4 Displaying Graphical Shapes 101
3.5 Graphics Structures 101
3.6 Choosing a Coordinate System 109
3.7 Getting Input from the Graphics Window 113
3.8 Comparing Visual and Numerical Information 114

Chapter 4 Decisions 127

4.1 The `if` Statement 128
4.2 Relational Operators 131
4.3 Input Validation 136
4.4 The `if/else` Statement 138
4.5 Multiple Alternatives 141
4.6 Nested Branches 149
4.7 Logical Operations 154
4.8 De Morgan's Law 159
4.9 Using Boolean Variables 160

Chapter 5 Functions 173

5.1 Functions as Black Boxes 174
5.2 Writing Functions 175
5.3 Function Comments 178
5.4 Return Values 181
5.5 Parameters 184
5.6 Side Effects 186
5.7 Procedures 187
5.8 Reference Parameters 189
5.9 Variable Scope and Global Variables 192
5.10 Stepwise Refinement 194
5.11 From Pseudocode to Code 195

5.12 Walkthroughs 203
5.13 Preconditions 207
5.14 Recursion 210

Chapter 6 Iteration 231

6.1 Simple Loops 232
6.2 Other Loop Statements 238
6.3 Common Loop Types 248
6.4 Simulations 259
6.5 Nested Loops 262
6.6 Block Scope 267

Chapter 7 Testing and Debugging 289

7.1 Unit Tests 290
7.2 Selecting Test Cases 292
7.3 Test Case Evaluation 294
7.4 Assertions 297
7.5 Program Traces 298
7.6 The Debugger 299
7.7 Strategies 310
7.8 Debugger Limitations 311

Chapter 8 Classes 321

8.1 Discovering Classes 322
8.2 Interfaces 323
8.3 Encapsulation 327
8.4 Member Functions 329
8.5 Default Constructors 332
8.6 Constructors with Parameters 334
8.7 Accessing Data Fields 338
8.8 Comparing Member Functions with Nonmember Functions 339
8.9 Object-Oriented Design 343

Chapter 9 Vectors and Matrices 371

9.1 Using Vectors to Collect Data Items 372
9.2 Vector Subscripts 376
9.3 Vector Parameters and Return Values 383
9.4 Simple Vector Algorithms 386
9.5 Parallel Vectors 390
9.6 Vectors as Object Data 394
9.7 Matrices 399

Chapter 10 Files 427

10.1 Reading and Writing Text Files 428
10.2 Example: Reading Input Data in a Graphics Program 429
10.3 Command Line Arguments 431
10.4 Random Access 436

Chapter 11 Modules 451

11.1 Separate Compilation 452
11.2 Designing Header Files 452
11.3 Breaking Up a Program into Multiple Source Files 457
11.4 Sharing Variables and Constants between Modules 463
11.5 Projects 466

Chapter 12 Algorithms 477

12.1 Selection Sort 478
12.2 Profiling the Selection Sort Algorithm 480
12.3 Analyzing the Performance of the Selection Sort Algorithm 482
12.4 Merge Sort 484
12.5 Analyzing the Merge Sort Algorithm 487
12.6 Searching 491

12.7 Binary Search 492
12.8 Searching and Sorting Real Data 495
12.9 The Efficiency of Recursion 497

Chapter 13 An Introduction to Data Structures 509

13.1 Vector-Based Linked Lists 510
13.2 Dynamic Memory Allocation 515
13.3 Linked Lists with Dynamic Memory Allocation 520
13.4 List Element Access with a Cursor 523
13.5 Binary Search Trees 535

Chapter 14 Inheritance and Polymorphism 553

14.1 Inheritance 554
14.2 Calling the Base-Class Constructor 557
14.3 Calling Base-Class Member Functions 558
14.4 Polymorphism 562
14.5 Virtual Functions 567
14.6 Class Relationships 575

Appendix A1 C++ Language Coding Guidelines 587

A1.1 Introduction 587
A1.2 Modules 588
A1.3 Functions 590
A1.4 Variables 591
A1.5 Constants 592
A1.6 Classes 593
A1.7 Control Flow 593
A1.8 Lexical Issues 594

Appendix A2 C++ Language Summary 599

A2.1 Types 599
A2.2 Variables 599
A2.3 Expressions 600
A2.4 Classes 603
A2.5 Functions 604
A2.6 Statements 605
A2.7 Pointers 608
A2.8 Mathematical Functions 609
A2.9 Strings 610
A2.10 Vectors and Matrices 611
A2.11 Input and Output 611
A2.12 Graphics 612
A2.13 Time 614
A2.14 Employee Records 614
A2.15 Life without ccc.h 614

Appendix A3 Moving from C++ to C 617

A3.1 Data Types and Variables 617
A3.2 Input and Output 618
A3.3 Functions 620
A3.4 Arrays 622
A3.5 Strings 627
A3.6 Structures 637
A3.7 Files 639
A3.8 Dynamic Memory Allocation 641

Glossary 643
Photocredits 653
Index 655

CHAPTER 1

Introduction

Objectives

- To understand the activity of programming
- To learn about the architecture of computers
- To learn about machine languages and higher-level programming languages
- To become familiar with your compiler
- To compile and run your first C++ program
- To recognize syntax and logic errors

1.1 What Is a Computer?

You have probably used a computer for work or fun. Many people use computers for everyday tasks such as balancing a checkbook or writing a term paper. Computers are good for such tasks. They can handle repetitive chores, such as totaling up numbers or placing words on a page, without getting bored or exhausted. More importantly, the computer presents you the checkbook or the term paper on the screen and lets you fix up mistakes easily. Computers make good game machines because they can play sequences of sounds and pictures, involving the human user in the process.

Actually, what makes all this possible is not just the computer. The computer must be programmed to perform these tasks. One program balances checkbooks; a different program, probably designed and constructed by a different company, processes words; and a third program plays a game. A computer itself is a machine that stores data (numbers, words, pictures), interacts with devices (the monitor screen, the sound system, the printer), and executes programs. Programs are sequences of instructions and decisions that the computer carries out to achieve a task.

Today's computer programs are so sophisticated that it is hard to believe that they are all composed of extremely primitive operations. A typical operation may be one of the following.

Put a red dot onto this screen position.

Send the letter A to the printer.

Get a number from this location in memory.

Add up these two numbers.

If this value is negative, continue the program at that instruction.

Only because a program contains a huge number of such operations, and because the computer can execute them at great speed, does the computer user have the illusion of smooth interaction.

The flexibility of a computer is quite an amazing phenomenon. The same machine can balance your checkbook, print your term paper, and play a game. In contrast, other machines carry out a much narrower range of tasks; a car drives, and a toaster toasts. Computers can carry out a wide range of tasks because they execute different programs, each of which directs the computer to work on a specific task.

1.2 What Is Programming?

A computer program tells a computer, in minute detail, the sequence of steps that are needed to fulfill a task. The act of designing and implementing these programs is called computer programming. In this course, you will learn how to program a computer—that is, how to direct the computer to execute tasks.