

PREFACE

Welcome — And Thank You!

Thank you for choosing *C++ For Artists: The Art, Philosophy, and Science of Object-Oriented Programming*. You have selected an excellent book to augment your C++ and object-oriented programming learning experience.

If you purchased this book because it is required for a course you may feel like you had no say in the matter. Paying for expensive college books feels a lot like having your arm twisted behind your back at the check-out counter. If it will make you feel better I will let you in on a secret. You bought a keeper.

If you are standing in the computer section of your favorite book store reading these lines and trying hard to decide if you should buy this book or pay the rent I say to you this: If you knew the stuff inside this book you could easily own your own place. The landlord can wait.

Target Audience

C++ For Artists targets the student who demands more from a C++ programming text book. What do I mean by student? A student is anyone who holds this book in their hands and by reading it expects to gain C++ and object-oriented programming knowledge. You may be a student enrolled in a high school, college, or university — or a practicing programmer seeking ways to expand your understanding of C++ and object-oriented programming. However you come to hold this book in your hands — you are my target audience.

Approach

C++ For Artists examines the topic of C++ and object-oriented programming from three unique perspectives.

First, programming is an art. It takes lots of skill (gained through study and training) and practice (gained from writing code) to succeed as a programmer. Talent separates the good programmers from the really great programmers. Just like some people have a knack for painting, some people have a knack for programming.

Second, object-oriented programmers can significantly benefit from a guiding philosophy. One that shows them how to tap their creativity, conquer challenges, and tame conceptual and physical complexity associated with large software systems.

Lastly, most programming students are not formally exposed to real-life, practical programming techniques and almost no object-oriented foundational theory during their tenure in the classroom.

These three perspectives: 1) programmer as artist, 2) creative approach philosophy, and 3) object-oriented programming theory, converge in *C++ For Artists* resulting in a truly unique programming text book.

Arrangement

The book is arranged into four parts: *Part I: The C++ Student Survival Guide*, *Part II: Language Fundamentals*, *Part III: Implementing Polymorphic Behavior*, and *Part IV: Intermediate Concepts*. Each part and its accompanying chapters are described in greater detail below.

Part I: The C++ Student Survival Guide

Part I: The C++ Student Survival Guide consists of four chapters designed to help you get a jump on your programming projects. The survival guide is meant to be referenced throughout your learning experience. The key features and discussion points of part I include:

- A discussion of the “flow”,
- A project approach strategy to be used to maintain a sense of progress when working on programming projects,
- A complete treatment on how to create C++ projects with two popular integrated development environments (IDEs) on Macintosh, Windows, and UNIX platforms,
- A step-by-step project walkthrough that applies the project approach strategy and development cycle to produce a complete working project.

Chapter 1: An Approach To The Art Of Programming

Chapter 1 begins with a discussion of the challenges you will face as you study C++ and object-oriented programming. It presents a project approach strategy specifically designed to help you maintain a sense of forward momentum when tackling your first programming projects. The chapter also presents a development methodology, a philosophical discussion of the concept of the “flow”, and practical advice on how to manage a programming project’s physical and conceptual complexity. I will show you how to use three important preprocessor directives: `#ifndef`, `#define`, and `#endif` to create separate header files. You may or may not be familiar with all the terms used in the chapter, especially those related to preprocessor directives and identifier naming, however, you are encouraged to return to the chapter as required. It serves to offer you a glimpse of things to come.

Chapter 2: Small Victories: Creating Projects With IDEs

Chapter 2 shows you step-by-step how to create C++ projects using two popular integrated development environments: Metrowerks CodeWarrior on the Macintosh, and Microsoft Visual C++ for the Windows platform. The focus of the chapter is the concept of the project and the steps required to create projects regardless of the IDE employed. If you prefer to use UNIX development tools this chapter also shows you how to use the make utility and how to create a makefile that can be used to compile, link, and manage multi-file projects.

Chapter 3: Project Walkthrough: An Extended Example

Chapter 3 takes you step-by-step through a complete programming project from specification to final implementation. Along the way you are shown how to apply the project approach strategy and the development cycle to arrive at an acceptable project solution. The `#ifndef`, `#define`, and `#endif` preprocessor directives are used to create safe header files that separate function interface declarations from function implementation code. If you are a novice student I do not expect you to fully comprehend all the material or programming techniques presented in this chapter, rather, the material serves as an excellent reference to which you will return periodically as you apply bits and pieces of this knowledge in your first programming projects.

Chapter 4: Computers, Programs, and Algorithms

Chapter 4 presents background information on computer hardware organization, memory systems, and algorithms. The emphasis is on understanding exactly what a program is from a computer and human perspective. I discuss the four phases of the program execution cycle, how program instructions are differentiated from ordinary data, and how memory is organized on common computer systems. I also talk about what makes a good and bad algorithm.

Part II: C++ Language Fundamentals

Part II presents a treatment of the core C++ programming language features and comprises chapters 5 through 13. This is a critical part of the book because it prepares you for further study of intermediate and advanced C++ and object-oriented concepts. The key features and discussion points of part II include:

- The unique ordering of the material. For instance, pointers are covered early so you will understand their use in other language constructs,
- Pointers are presented as a dialog between a superhero named C++ Man and a confused student named Perplexed One,
- Emphasis on multi-file projects,
- Lots of targeted code examples to reinforce key lecture points,
- Successive chapters build upon knowledge gained from the previous chapter,
- In-depth coverage of tricky concepts normally glossed over or avoided in ordinary C++ texts.

Chapter 5: Simple Programs

Chapter 5 shows you how to write simple C++ programs using fundamental data types and simple expressions. I give examples of how to use all the C++ operators, how to create local and multi-file variables and constants, and show you how you can limit a variable's scope to one file. You will learn how to write two versions of the main() function and how to call functions upon program exit.

Chapter 6: Controlling The Flow Of Program Execution

Chapter 6 moves beyond simple programs and shows you how to control the flow of program execution by using if, if-else, switch, for, while, and do-while statements. Many source code examples and diagrams are used to illustrate how control flow statements are written. The chapter includes a discussion of statements, null statements, and compound statements. I also show you how to write nested if, for, and while statements, and how to write loops that will repeat until explicitly exited.

Chapter 7: Pointers And References

Chapter 7 uses a short story to simplify the complex topic of pointers and references. Perplexed One is a student who falls asleep in class and is awakened by the arrival of C++ Man. C++ Man then helps Perplexed One by answering questions and giving examples of how to declare and use pointers.

Chapter 8: Arrays

Chapter 8 builds upon chapter 7 and shows the relationship between pointers and arrays. The chapter continues by showing you how to build single and multi-dimensional static and dynamic arrays. Lots of code examples and diagrams help you visualize how arrays are declared, initialized, and used in programs.

Chapter 9: Functions

Chapter 9 builds upon chapter 8 and shows you how to write complex functions that can pass arguments by value and by reference. The emphasis is on writing highly cohesive functions that are minimally coupled to other program elements. Header files are used to separate function declaration (interface) from definition (implementation). To support the creation of header files I review and discuss the three important preprocessor directives: #ifndef, #define, and #endif. Other topics covered include: function variable scoping, static function variables, passing arrays to functions, passing multi-dimensional arrays to functions, returning pointers from functions, how to avoid dangling references, function overloading, recursion, function pointers, and call back functions.

Chapter 10: Toward Problem Abstraction: Creating New Data Types

Chapter 10 shows you how to create type synonyms and new data types using type definitions, enumerated types, structures, and classes. The emphasis is on problem abstraction and how it is used to map a real world problem to a set of supporting data structures that can be used in a program. Structures are compared to classes and the notion of object-oriented programming is introduced. The class declaration is discussed as it relates to the structure declaration

and how the notions of procedural and object-oriented programming differ from each other.

Chapter 11: Dissecting Classes

Chapter 11 continues the discussion of classes and how they work. It introduces the UML class diagram and uses UML class diagrams to illustrate static class relationships. The special member functions are thoroughly discussed. These include the constructor, destructor, copy constructor, and copy assignment operator. A brief introduction to the orthodox canonical class form is given in preparation for a deeper treatment of the subject in chapter 17. Other topics include data encapsulation, member functions and attributes, access specifiers, member function overloading, and how to separate class interface from implementation.

Chapter 12: Compositional Design

Chapter 12 builds upon chapter 11 and shows you how to build complex class types using simple and complex aggregation. The UML class diagram is extended to model simple and composite aggregate class relationships. The UML sequence diagram is also introduced to illustrate interobject message passing. Other topics discussed include: managing physical complexity, the use of pointers and references to build simple and complex aggregate classes, and how to properly use constructors and destructors in aggregate classes. The chapter concludes with a complex aggregation example.

Chapter 13: Extending Class Functionality Through Inheritance

Chapter 13 introduces the topic of inheritance and shows you how to extend class behavior through subclassing and subtyping. UML is used to illustrate simple and complex inheritance hierarchies. The compositional design techniques discussed in chapter 12 are combined with inheritance design concepts to provide you with a powerful arsenal of object-oriented design tools. The access specifiers public, protected, and private are discussed in the context of inheritance. Other topics covered include: virtual functions, function hiding, function overloading, pure virtual functions, abstract classes, abstract base classes, multiple inheritance, and virtual inheritance. The chapter includes a complex navy fleet simulation example that illustrates the use of inheritance and compositional design.

Part III: Implementing Polymorphic Behavior

Part III gives special coverage to the three types of polymorphic behavior: ad hoc (operator overloading), static (templates), and dynamic (base class pointers to derived class objects). Success as a C++ programmer demands a thorough understanding of these concepts. Key features and discussion points of part III include:

- In-depth treatment of ad-hoc, static, and dynamic polymorphism and how each type of polymorphic behavior is achieved using the C++ language,
- An example of how to overload almost every operator in the C++ language,
- How to overload the iostream operators to tailor them to your class needs,
- How to think about and apply the notion of polymorphic behavior in your application designs,
- How to write generic code using templates,
- How to use multiple place holders in template classes and functions,
- How to use the special template definition syntax to explicitly specify template parameter types,
- How to design with dynamic polymorphic behavior in mind.

Chapter 14: Ad Hoc Polymorphism: Operator Overloading

Chapter 14 is devoted to operator overloading. It builds upon the concepts of function overloading and shows you how to overload nearly every operator in the C++ language complete with examples of their use. A complete table of overloadable operators is included along with a discussion of how to overload the iostream operators to tailor them to your class needs.

Chapter 15: Static Polymorphism: Templates

Chapter 15 shows you how to write generic code using templates. It shows you how to replace overloaded functions with template functions and how to use template functions your programs. The chapter also shows you how to

use the special template definition syntax to explicitly specify template parameter types. A brief overview of the C++ standard template library (STL) is offered along with a discussion of STL containers, iterators, and algorithms.

Chapter 16: Dynamic Polymorphism: Object-Oriented Programming

Chapter 16 reinforces and builds upon concepts introduced in chapter 13. The focus is on the C++ language constructs required to write truly object-oriented programs. Topics discussed in depth include: employing pure virtual functions to create abstract base classes, how to use abstract base classes to specify the interface to derived classes, and what behavior to expect when using dynamic polymorphic programming techniques. The engine component aggregate class created in chapter 12 is revisited and redesigned to employ dynamic polymorphic behavior.

Part IV: Intermediate Concepts

Part IV consists of four chapters and builds upon the concepts and material presented in the preceding three parts. Key features and discussion points of part IV include:

- How to write well-behaved, context-minded classes using the orthodox canonical class form,
- How to use legacy C code libraries in your C++ applications,
- How to use the Java Native Interface (JNI) to write C++ functions that can be called from Java applications,
- How to use assembly language in C++ programs,
- Coverage of three important object-oriented design concepts to include the Liskov substitution principle and Meyer design by contract programming, the open-closed principle, and the dependency inversion principle,
- How to use a UML design tool to assist in the design and implementation of complex applications,
- How to use a UML tool to reverse engineer existing C++ code.

Chapter 17: Well-Behaved Objects: The Orthodox Canonical Class Form

Chapter 17 presents an in-depth discussion of the orthodox canonical class form (OCCF) to write well-behaved, context-minded classes. Keeping the OCCF in mind when you design and write classes forces you to consider how those classes will be used in an application. The class's possible uses or usage contexts will guide you in your choice of which operators to overload to insure your class objects exhibit predictable and acceptable behavior.

Chapter 18: Mixed Language Programming

Chapter 18 shows you how to use C++ with C, assembly, and Java. Topics covered include: using the extern keyword to prevent C++ name mangling, the Java Native Interface (JNI) and how to write C++ functions that can be called from Java programs, how to use inline assembly code in C++ programs using the asm keyword, and how to link to object modules created in assembly language.

Chapter 19: Three Design Principles

Chapter 19 presents and discusses three important object-oriented design principles: the Liskov substitution principle, the open-closed principle, and the dependency inversion principle. Bertrand Meyer's design by contract programming is discussed in relation to the Liskov substitution principle.

Chapter 20: Using A UML Modeling Tool

Chapter 20 discusses the importance of using a UML design tool to assist in the application design process. The featured UML tool is Embarcadero Technologies's Describe™. The chapter focuses on only a few of Describe's many features: UML use-case, class, and sequence diagram creation, how to link diagram objects to other diagrams, how to generate code from class diagrams, how to reverse engineer existing C++ code, and how to generate comprehensive web-based project reports.

Pedagogy

Chapter Layout

Each chapter takes the following structure:

- Learning Objectives
- Introduction
- Content
- Quick Reviews
- Summary
- Skill Building Exercises
- Suggested Projects
- Self Test Questions
- References
- Notes

Learning Objectives

Each chapter begins with a set of learning objects. The learning objectives represent the knowledge gained by reading the chapter material and completing the skill building exercises, suggested projects, and self test questions.

Introduction

The introduction motivates you to read the chapter content.

Content

The chapter content represents the core material. Core material is presented in sections and sub-sections.

Quick Reviews

The main points of each level 1 section are summarized in a quick review section.

Summary

The summary section summarizes the chapter material

Skill-Building Exercises

Skill building exercises are small programming or other activities intended to strengthen your C++ programming capabilities in a particular area. They could be considered focused micro-projects.

Suggested Projects

Suggested projects require the application of a combination of all knowledge and skills learned up to and including the current chapter to complete. Suggested projects offer varying degrees of difficulty.

Self-Test Questions

Self test questions test your comprehension on material presented in the current chapter. Self test questions are directly related to the chapter learning objectives. Answers to self test questions appear in appendix C.

References

All references used in preparing chapter material are listed in the references section.

Notes

Note taking space.

CD-ROM

The CD-ROM contains the following goodies:

- PDF edition of C++ For Artists,
- Adobe Acrobat™ Reader version 6 for Windows and Macintosh
- Demo version of Embarcadero Technologies Describe™ UML modeling tool,
- Full working copy of ObjectPlant™ UML modeling tool for the Macintosh™,
- Open source C++ compiler tools,
- All source code example files used throughout the text organized by chapter,
- Metrowerks CodeWarrior projects,
- Links to commercial C++ development tool vendors.

SupportSite™ Website

The C++ For Artists SupportSite™ is located at [<http://pulpfreepress.com/SupportSites/C++ForArtists/>]. The support site includes source code, links to C++ compiler and UML tool vendors, and corrections and updates to the text.

Problem Reporting

Although every possible effort was made to produce a work of superior quality some mistakes will no doubt go undetected. All typos, misspellings, inconsistencies, or other problems found in C++ For Artists are mine and mine alone. To report a problem or issue with the text please contact me directly at rick@pulpfreepress.com or report the problem via the C++ For Artists SupportSite™. I will happily acknowledge your assistance in the improvement of this book both online and in subsequent editions.

Acknowledgements

C++ For Artists was made possible by the hard work and support of many talented people and companies. Some friends contributed unknowingly in unexpected ways.

I would first like to thank *Harish Ruchandani* and *Tracy Millman*, my former colleagues at Booz | Allen | Hamilton, for patiently listening to my ideas about writing this book and for providing critical comment on early versions of several chapters.

Many thanks to my good friend *Jose Pi* for many great mornings spent surfing California waves, and to *Michael Leahy*, a merchant mariner of the highest caliber, for letting me drive his Ferrari with no strings attached.

I would like to thank *Anke Braun*, *Thayne Conrad*, and *Petra Rector* of Prentice-Hall for entertaining my proposal and trying to fit C++ For Artists into the Prentice-Hall product line. Thanks also go to *Jim Leisy* of Franklin, Beedle & Associates, Inc., for seeing the merit in this work.

Special thanks go to the reviewers employed by Prentice-Hall who provided invaluable critical comment on chapters 1 through 13. They include: *John Godel*, *James Huddleston*, *Dr. Samuel Kohn*, and *Ms. Anne B. Horton*. C++ For Artists is significantly improved by their attention to detail.

Independent reviewers of different portions of the text include *Ken Stern* and *Brendan Richards* of SAIC. It is truly a pleasure working with such talented people.

I want to thank *Apple™ Computer Inc.*, for providing product images of the PowerMac™, *Embarcadero Technologies, Inc.*, for granting me a full-use license of Describe™, and *Michael Archtadeous* for working in the trenches to produce ObjectPlant™.

Lastly, without the fathomless patience of Coralie Miller, an amazing woman, this book would simply not exist.

A handwritten signature in black ink, consisting of a large, stylized 'R' followed by a smaller 'M' and a long, sweeping horizontal line extending to the right.

Rick Miller
Falls Church, Virginia
7 July 2003

