

Using C++

Using C++

Bruce Eckel



Osborne McGraw-Hill

Berkeley New York St. Louis San Francisco
Auckland Bogotá Hamburg London Madrid
Mexico City Milan Montreal New Delhi Panama City
Paris São Paulo Singapore Sydney
Tokyo Toronto

9450014

Osborne McGraw-Hill
2600 Tenth Street
Berkeley, California 94710
U.S.A.

2287/38

For information on translations and book distributors outside of the U.S.A., please write to Osborne McGraw-Hill at the above address.

A complete list of trademarks appears on page 607.
Reprinted with coreections, April, 1990.

Using C++

Copyright © 1989 by McGraw-Hill, Inc. All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

34567890 DOC/DOC 99876543210

ISBN 0-07-881522-3

Acquisitions Editor: Jeff Pepper
Technical Reviewer: Ron Burk
Copy Editor: Leslie Tilley
Proofreaders: Barbara Conway, Julie Anjos
Word Processor: Judy Koplan
Composition: Bonnie Bozorg
Production Supervisor: Kevin Shafer

This book was produced using Ventura Publisher Version 2.

Information has been obtained by Osborne McGraw-Hill from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, Osborne McGraw-Hill, or others, Osborne McGraw-Hill does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.

2:00528

Acknowledgments

There were several technical readers for this book. Foremost was Ron Burk of Burk Labs in Seattle. Ron's thorough knowledge and acid pen (ouch!) are greatly responsible for making this book accurate. John Carolan of Glockenspiel in Dublin also read many of the chapters and contributed comments. Bjarne Stroustrup read Chapter 11 and answered many questions about the language by electronic mail—questions I don't think I could have answered any other way. Other technical readers were Walter Bright (Zortech) and Brian McElhinney.

I received help with the style from my friend and editor at Micro Cornucopia Magazine, Gary Entsminger, and the staff at Osborne/McGraw-Hill, including acquisitions editor Jeff Pepper and associate editor Judith Brown.

Emotional support came primarily from Melinda. Comraderie was provided by Carl Haflinger, Mary Nerini, Marilyn Cvitanic, Tim Hohn, Cheri Singer and Sam, Mark Bennett Western, the Moelter Family, the Robbins Family, Allison Brody, Michael Wilk, Dave and Sue Renschler, Dave Mayer, Patrick Conley, Joseph McIssac, Brad Jerbic, Todd, Lynn, Jim and Brice. Larry Fogg contributed the occasional Fractal Thought Pattern.

A lot of the incentive for this book came from my friend Daniel Will-Harris, with whom I seem to have been competing in everything since the first day in junior high school, when we met. We now both have two computer books to our names. If he (and Toni, of course) get a screenplay produced, I could be in trouble. However, that bet about who would be tallest will always be my ace (you still owe me thousands of dollars for that one, DWH!).

Preface

It seems that the most creative and innovative software is developed by small teams of one or two people. As software projects have become larger and more sophisticated, development teams have grown larger in an attempt to cope with the complexity and deadlines. We have seen these teams, and the companies that manage them, lose the clarity of their vision about a project, or simply lose the ability to implement it in a timely manner. The project either fails to materialize, or the delivery date slips, and slips, and slips...

C++ is an object-oriented extension to the C programming language. An object-oriented language allows, among other things, the complexity of a program to be hidden. Most languages come with built-in data types and ways for you to use those data types (add them, pass them around, print them, and so on). C++ and other object-oriented languages allow you to *define your own data types and the ways to use those types*. This is a powerful ability, but it also opens up many new questions (How will you pass objects around? Will you add them? If so, how? How will an object handle errors?). Most users of a language don't need to think about these details (and users of predefined data types in C++ don't need to, either) but when you begin defining your own types each of these questions must be answered. This book serves as a guide to answering those questions.

C++ improves productivity. It does this not by imposing a structure that you must follow if you want the program to come out "right," but by creating a framework in which building an easy-to-read, robust, maintainable, extensible program is the most natural path. This is true whether one is experimenting or creating production code—in fact, you will find that much of the code you create while experimenting survives into production.

The size of the problem a team can manage is based on their abilities and the sophistication of their tools. One of the great benefits I see in C++ is that it puts the power needed to build large, complex projects back in the hands of the one- or two-person team. This means we may start seeing the kind of wild innovation that was so refreshing in the early days of the

computer revolution (which I start counting from the time when computers got cheap enough that you weren't forced to do "serious" work on them all the time—people aren't creative when they have to be serious). C++ certainly isn't limited to small projects, however. The language provides a way to communicate interface specifications between members of a large programming team and to enforce the correctness of those interfaces at compile-time. In addition, the ability to easily create and maintain large libraries of useful tools will be a benefit to one-person projects as well as very large teams.

There are numerous advantages to using C++. Some of the reasons the language was designed are

- To simplify the building and use of libraries.
- To allow code re-use. If a library function doesn't suit your needs, you can easily modify a portion of it without understanding the whole thing—whether or not you have access to the source code.
- To improve "maintainability" of code. Since the language supports object-oriented design, code is generally much easier to understand, fix, and modify (someone with a strong bent can still write bad code, however.) C++ fosters systems that can be designed for extensibility.

Many people who learn the language discover a more mysterious benefit. Once a program compiles, it often seems to "just work right the first time." Although there is no way to measure or verify this benefit, it is probably due to the stronger type checking C++ has over C, and the structure even the most avid hackers are seduced into using when creating new data types.

C++ is the first object-oriented language with efficiency as one of its primary goals. Inefficiency and lack of compile-time error checking has held object-oriented languages back from being practical in many production programming situations.

I see the language from an individual viewpoint. One of my favorite things about C++ is that it makes me think about programming in ways I've never considered before. Most commonly used procedural languages make you think about programming; C++ helps you think about problem-solving.

I've had the same experience with C++ as I had when I learned calculus. Before I learned calculus, algebra and trigonometry were interesting, but only marginally useful. I didn't remember them very well. When I learned

calculus, a lot of things about algebra and trigonometry became very clear and I quickly learned their intricacies because I had to use them, without thinking about it, on a daily basis. The same thing happened with C++. C++, like calculus, added a new dimension to my thinking. From this new perspective, I could easily see the reasoning behind most of the features in my previous framework of thinking (which was C and Pascal). My knowledge of C has improved greatly since I started programming in C++ in 1987. Except for operator precedence—I've never found a way to make that stick.

If you can shake yourself free from the mindset of "fitting the problem to the computer," I think that you'll find the naturalness of "sending messages to objects" to be a very comfortable and powerful way of thinking about problems.

The Organization of this Book

This book assumes you already know another programming language. It would be easiest, of course, to assume that language is C. However, many people haven't had any incentive to learn C—their own programming language (probably BASIC or Pascal) has been satisfactory—until now. C++ offers enough real benefits over traditional languages that large numbers of programmers will want to learn it.

The book is divided into two parts. After introducing object-oriented languages, Part One (including Chapters 1-4) presents the syntax of ANSI C and C++, covering the features that are common between the two languages, describing the extensions that C++ adds to ANSI C, and especially flagging the places where they are different. This section focuses on more than just the syntax of the languages, however. When you begin writing larger programs in C++, you inevitably create many files that must be managed and compiled properly to build the project. Part One covers the proper construction of multifile projects, emphasizing header files and the **make** utility. All the programs in each chapter, in fact, are created with the **makefile**, which is placed at the end of each chapter.

Part Two covers object-oriented and advanced programming in C++. While Part One shows you what the language features *are*, Part Two shows you how to use them. Chapter 5 shows you how to "overload" operators and functions. Operator overloading means you can give an operator like + or - a special meaning when used with a new data type you create. Function overloading means you can create several meanings for the same function identifier, depending on the argument list.

Chapter 6 shows you how to create objects at run time, for situations when you don't know at compile-time how many objects you will need or what their lifetimes will be. Creating objects at run time is an extremely powerful feature of C++.

Chapter 7 demonstrates how code can be re-used in C++, both by using objects inside of other objects, and through the mechanism of *inheritance*. Chapter 8 shows you how to use inheritance to build extensible programs. Once you've built an extensible program, you or someone else can easily add features without ripping the existing code apart. The ability to build extensible programs will save programmers a lot of time (and software companies a lot of money).

Chapters 9-11 cover advanced topics. Chapter 9 considers the somewhat thorny problems of passing object arguments and returning object values. Much emphasis is placed on the copy-constructor `X(X&)`, which often causes new C++ programmers a lot of grief. Chapter 10 has several complete examples, which provide you with some projects to sink your teeth into. Finally, Chapter 11 covers the latest release of the C++ language from Bell Labs, AT&T release 2.0.

The book has three appendices, each of which is a programming project. Appendix A is a small graphics application that suggests a framework for building a CAD system in C++. Appendix B is a mathematical matrix manipulation package. Appendix C is a text windowing class. Appendices A and C use library functions specific to a certain compiler (Zortech C++), but these functions are hidden in the class methods and may easily be changed. All the rest of the examples in the book should compile under any C++ implementation.

Throughout the book, I've tried to use interesting examples. I have tried to avoid repeating the typical textbook examples in favor of programs that have either been useful or fascinating. One of the more difficult problems in teaching this language seems to be the order in which concepts are introduced. Great efforts have been made to ensure a concept is introduced before it is used in an example, or at least to tell you where a concept is explained, in the few cases where they are used prematurely.

A Note About Compilers

While I was writing this book, I used two different implementations of C++ on the PC: Glockenspiel C++ and Zortech C++ (I have also used AT&T **cfront**, Gnu C++, and Oregon Software C++ on the Sun 3, but not for this book). At the time, both products contained discrepancies with the C++

language. Between the two, however, I could find a way to compile all the examples of correct C++. By the time you read this, many or all of the discrepancies should be fixed. If you are doing serious development work, especially if you want to port it to other platforms, I highly recommend that you acquire more than one implementation of C++. Just because the compiler complains about something doesn't mean it's a bug; and just because code compiles on one implementation doesn't mean it is legal C++ or will work with other compilers. Someday problems like this may be fixed. Until then, you will get surprises unless you constantly test your code for portability.

Source Code Disk

For your convenience, you can get all the sources listed in this book on a 5-1/4" DS/DD IBM PC-formatted diskette. Each chapter has its own subdirectory and the **makefile**, which is at the end of every chapter, so all you need to do to compile the code is move into the subdirectory and type **make**. There are some further examples; tools I used in the preparation of this book, programs I used to investigate compiler features, and (space allowing) anything else I could add that didn't make it into the book.

The listings in the book have all been compiled. After testing, the source code was automatically put into book format, so human hands didn't have a chance to introduce errors. Sometimes, however, logic errors are discovered, or a compiler bug that allows incorrect syntax to pass through unflagged is fixed. If you discover errors like this, please mail them to the publisher (the address is shown on the copyright page). They will be fixed in the book at the earliest possibility, and the source-code disk will be updated immediately. If you have trouble with a listing, chances are it will be fixed on the source-code disk.

The disk is available postpaid for \$25 from

Revolution2

501 N. 36th Street, Suite 163

Seattle, WA 98103

Checks only, please. Foreign orders please add \$7, and use a check in U.S. funds drawn on a U.S. bank or an international postal money order. See the form at the end of this preface for details.

My firm, Revolution2, offers C++ consulting and on-site training. For more information, write to the address given for source code disk orders. You can get the company's phone number from the *C Gazette*, where I am the C++ editor and columnist, by calling (213) 473-7414. This is also the number to call if you want a subscription to the magazine. I can be reached electronically on BIX as Beckel, on Compuserve as 72070,3256 or on the internet as 72070.3256@compuserve.com.

Source-Code Disk Order Form

Please send me _____ copies of *Using C++ Source Code Disk* (plus additional projects). I have enclosed a check for \$25 for each disk. (Foreign orders please add \$7 for shipping and handling, and use a check in US funds drawn on a US bank).

Name _____

Company _____

Address _____

City _____

State _____

Zip _____

Telephone _____

IBM PC Diskette size (check one) 5 $\frac{1}{4}$ " _____ 3 $\frac{1}{2}$ " _____

Send to:

Revolution2
501 N. 36th Street, Suite 163
Seattle, WA 98103

Please send me information on Revolution2 consulting & training _____

Please include any comments you wish the author to read.

Osborne/McGraw-Hill assumes NO responsibility for this offer. This is solely an offer of the author, Bruce Eckel, and not of Osborne/McGraw-Hill.

Contents

Preface	xv
Part One	
Introduction to Object-Oriented Languages	1
1 Introduction to Object-Oriented Languages	3
Basics and Conventions	3
Why Use an Object-Oriented Language?	4
The Evolution of Object-Oriented Languages	5
Support for Experiments and Structure	7
The Process of Language Translation	8
Why You Need Objects	11
The Advantages of Inheritance	15
The Definition of An Object-Oriented Language	18
C++ Versus Smalltalk	19
Summary	20
2 Using Predefined Classes	21
Tools for Separate Compilation	22
Your First C++ Program	29
make—an Essential Tool for Separate Compilation	35

More About Streams	37
Controlling Execution in C and C++	41
Introduction to C and C++ Operators	49
Using Standard I/O for Easy File Handling	51
Utility Programs Using Streams and Standard I/O	55
Makefile for Chapter Examples	58
Summary	60
3 Creating Classes with C++	61
Introduction to C++ Data	61
Scoping	64
Specifying Storage Allocation	67
C and C++ Operators and Their Use	75
Creating Functions in C and C++	86
Unique Features of C++ Functions	90
C++ Function Overloading	92
The class: Defining Boundaries	95
The Header File	104
Defining class Member Functions	111
Other class-like Items	118
Debugging Hints	124
Makefile for Chapter Example	130
4 Pointers and References	133
Addresses Are Just Like Mailboxes	133
Pointers	134
Using Pointers and Addresses	138
Variable Pointers	141
Array Pointers	145
The Size of a Pointer (Memory Models)	153
Function Addresses	161
Examples of Pointer Use	168
The Reference Feature	177
When to Use References	190
Makefile for Chapter Examples	194

Part Two	
Object-Oriented Programming with C++	197
5 Overloading Functions and Operators	199
The Syntax of Operator Overloading	201
Examples of Operator Overloading	201
Creating Your Own Type	
Conversion Operators	217
Example: Creating Your Own	
Stream Functions	228
Selecting Friend or Member Functions	
for Operator Overloading	231
Function Overloading	235
Makefile for Chapter Examples	238
6 Creating Objects at Run Time	239
The Bias of Some Popular Languages	239
Dynamic Object Creation	241
Arbitrarily Sized Objects	255
The Mechanics of Dynamic Object Creation	267
Changing the Behavior of new and delete	274
Pitfalls in Dynamic Memory Allocation	280
Using References with Dynamic	
Memory Allocation	283
Assigning to this in a Constructor	283
Makefile for Chapter Examples	291
7 Reusing Code in C++	293
Reusing Code with Member Objects	295
Reusing Code with Inheritance	304
Storing Objects on Disk	326
A List That Can Save and Retrieve Itself	338
Accessing Elements of the Base Class	347
Makefile for Chapter Examples	351

8	Writing Extensible Programs in C++	355¹
	Coding the transportation Hierarchy in C++	357
	Virtual Functions	363
	An Extensible Menu System	371
	Abstract Classes	389
	Simulation Using Abstract Classes	396
	Makefile for Chapter Examples	404
9	Arguments and Return Values	409
	Passing by Name, Value, or Reference	410
	Hidden Activities	412
	The Copy-Constructor X(X&)	415
	Object Assignment	419
	An Example of Deep Copy: class matrix	422
	Tracing the Creation and Destruction of Objects	429
	Returning Objects with *new	436
	Makefile for Chapter Examples	442
10	Complete Examples	443
	Command-Line Arguments, Flags, and Files	443
	A Program to List Files	453
	TAWK: A Simple Database Interpreter	458
	A Clock-Based Control System	482
11	C++ Release 2.0	509
	Continuing Incompatibilities with ANSI C	510
	Multiple Inheritance	517
	Type-Safe Linkage	522
	The Copy-Constructor and operator=()	
	Can Be Created by the Compiler	526
	Operator new and delete Can Be Overloaded	528
	Global new and delete	531
	New Operators Can Be Overloaded	533
	Pointers to Members	536
	New Qualifiers for Member Functions	538
	New Forms of Initialization	541

Parameterized Types	547	
Exception Handling	549	
Miscellaneous Changes	551	
The Order of Initialization of Static Objects	556	
A MicroCAD		557
The MicroCAD Code	557	
Features to Add as an Exercise	575	
Makefile for MicroCAD	576	
B The Matrix Class		579
Standard Matrix Files	580	
Speed Improvements	581	
Code for the matrix class	581	
Makefile for matrix class	595	
C Windows		597
Code for Close Window	598	
Improvements You Could Add	605	
Makefile for Appendix C	605	
Index		609

Introduction to Object-Oriented Languages

Part One