Borland® C++ 3 Object-Oriented Programming

15

Borland® C++ 3 Object-Oriented Programming

Ted Faison

A Division of Prentice Hall Computer Publishing
11711 North College, Carmel, Indiana 46032 USA

Preface

Object-oriented programming (OOP) is a hot topic today. To many, C++ itself is synonymous with OOP, much the way many people implicitly associate LISP programming with artificial intelligence. The truth is, OOP is not so much a consequence of this or that language, but rather the result of the particular methods used. It is entirely possible to develop OOP applications with languages such as Pascal, Ada, BASIC, and assembly language, albeit with increasing difficulty.

What this book emphasizes is the OOP aspect of C++, using a particular implementation—Borland C++—as the vehicle. Given the recent explosion of interest in Microsoft Windows programming, the book covers selected topics of Windows programming, but only within the framework of OOP. The reader will learn to understand and reason in OOP terms when writing Windows applications, developing a methodology different from the traditional one presented in the Microsoft Windows Software Development Kit.

This is a "hands-on" book, because there are frequent programming examples and projects throughout. All sample programs and all code that appears with a listing number can be loaded and compiled immediately from the companion disk to test the various features of OOP shown. Much attention was given to such practical issues as functionality and efficiency. The reader is assumed to have experience with the C programing language. A minimum of two years is recommended. Advanced users will also find interesting material toward the end of each chapter.

Acknowledgments

A book like this is useful only if the information it contains is accurate, but it is very difficult to guarantee absolute correctness. I would like to thank all the people who provided corrections, suggestions, and input for this book. C++ streams are a particularly detailed field, and I am grateful to Lori Benner of Borland International for checking selected portions of Chapter 6. I also thank Borland's Nan Borreson, Phil Rose, Pete Becker, and Sydney Markowitz, among the many others, for their cooperation and help. Last, but certainly not least, I thank my editor Greg Croy for being so helpful, friendly, and professional during the many months it took to write this book.

Introduction

During the 1980s, C emerged as one of the world's premier and universal programming languages. It made it possible and efficient to write code that was portable to a wide class of computers. Software could be written faster and projects on the average grew in size. With size came complexity, leading to increased development times. Today development time and effort for software is a major issue in many companies. AT&T developed the C++ language as an extension of ANSI C, in an attempt to bring many of the advantages of object-oriented programming to the world of C without losing the many desirable features—such as simplicity and runtime efficiency—that made C so popular.

C++ was developed to make programming easier. To make this possible, the language had to be more complex than its predecessor. All the added features of C++ are aimed at reducing levels of difficulty. Obviously, the mere adoption of C++ doesn't automatically guarantee better or simpler software. To reap the benefits of C++, you must adopt a new programming methodology commonly referred to as *object-oriented programming*, or *OOP*.

Why Object-Oriented Programming?

Several years ago computer science researchers noted that programmers can write and debug pretty much the same amount of code no matter what language they use. The amount of work is roughly the same, but the results are not. Writing 100 lines of code in C is about as difficult as writing 100 lines of code in assembly language, but the C code accomplishes much more. With this in mind, researchers sought to develop higher-level languages that multiply the power of a single programmer, thus reducing project development time and costs.

In the 1970s, the concept of the *object* became popular among programming language researchers. An object is a collection of code and data designed to emulate a physical or abstract entity. Objects are efficient as programming items for two main reasons: They represent a direct abstraction of commonly used items, and they hide most of their implementation complexity from their users. The first objects developed were those most closely associated with computers, such as `Integer`, `Array`, and `Stack`. Some languages (such as Smalltalk) were designed as orthodox languages in which everything is defined as an object.

Object-oriented programming is a methodology that gives great importance to relationships between objects rather than implementation details. Relationships are ties between objects and are usually developed through genealogical trees in which new object types are developed from others. Hiding the implementation of an object results in the user being more concerned with an object's relation to the rest of the system rather than how an object's behaviors are implemented. This distinction is important and represents a fundamental departure from earlier "imperative" languages (such as C) in which functions and function calls were the center of activity.

In C++, few objects are part of the language itself. The burden and responsibility for devising objects is on the user. Borland C++ is bundled with a number of object types, but to make any real use of the language requires developing many more types. The power of OOP is exploited if groups of interrelated object types are developed. These groups are usually called *class hierarchies*. Developing these class hierarchies is a central activity in OOP.

The Structure of This Book

This book describes the new methodology required to develop class hierarchies based on Borland C++ 3.0, using the object types furnished by Borland International. Before doing so, the book covers the basic features of C++. The main OOP features of C++ are introduced in separate chapters.

This book is divided into two parts. The first part, "Object-Oriented Programming with Borland C++," describes the C++ language in general, with particular attention to Borland C++ and the features that make it an object-oriented language. This book is not intended to be a complete reference on Borland C++, but it shows how to use the language features in an object-oriented sense.

The second part of the book, "Developing Windows and DOS Applications," beginning with Chapter 8, shows how to use Borland C++'s class libraries and application development tools. In the 1990s, graphical interfaces are expected to supplant all other interfaces, with Windows being the premier GUI on DOS machines. Anticipating this, Borland has developed two application frameworks for its C++ 3.0 compiler to facilitate writing Windows or DOS programs. Much attention is given throughout the book to the program examples, which also are available in source code on the companion disk. Each of the examples has been tested and can be compiled and tried immediately.

The structure of this book is somewhat unusual. Although on the surface it is straightforward, in reality you sometimes are referred to material in later chapters. This is because I chose to describe C++ systematically rather than gradually. This makes it easier to find information in the book. For example, the section on class destructors in Chapter 2 has all the information on destructors, citing `virtual` destructors, even though `virtual` functions are described in detail only in Chapter 5. This order of presentation differs from that of most other C++ books; however, I believe the advantages outweigh the disadvantages.

Book Description

This book deals with C++ programming in general and Borland C++ 3.0 in particular. Frequently I refer to C++ rather than Borland C++ when a specific topic is general to the proposed ANSI C++ standard. Because Borland C++ is essentially a superset of AT&T C++ release 2.1, the distinction is necessary.

Chapter 1, "Basics," summarizes the main constructs of Borland C++ without making any formal definitions or presentations, giving space to topics in which C++ differs from ANSI C. Although C programs can be compiled with a C++ compiler, it is not true that C++ always uses the same techniques as C programs. Some C features considered obsolete in C++ are pointed out.

Chapter 2, "Objects and Classes," is the real beginning of the object-oriented extensions to ANSI C, introducing the new concepts of objects and classes. This chapter shows how code and data are used together to build an object, how objects are used, and what properties they have.

Chapter 3, "Inheritance," illustrates how objects can be built starting with other objects rather than from scratch. This enables the objects to inherit characteristics from the parent classes, reducing the amount of coding and debugging necessary to accomplish a task. Inheritance allows classes to be used repeatedly as *black boxes*, increasing programmer productivity. Both single inheritance and multiple inheritance are discussed.

Chapter 4, "Overloading," deals with function and operator overloading. Experienced programmers may yawn initially here, but don't even *think* of skipping this chapter. Overloading is an important characteristic that allows different classes to use a uniform notation for actions that are conceptually similar. This is another C++ simplification that comes to the aid of the programmer, helping you manage large projects better.

Chapter 5, "Polymorphism," covers one of the most touted features of C++. Polymorphism is described and shown as a concrete way to simplify programming through the use of `virtual` functions. Advantages and disadvantages of `virtual` functions are shown, including explanations of the runtime features of virtual functions.

Chapter 6, "Streams," deals with input and output (I/O). All programs have to produce results to be useful, so they must have a means for outputting information. In general, programs need both input and output. Chapter 6 describes input and output in terms of the new C++ constructs of *streams*. I/O streams are described for both files and hardware devices. The concept of the stream is also applied to in-memory operations.

Chapter 7, "The Container Class Library," is specific to Borland C++ 3.0 and doesn't apply to other compilers. The container class library furnished by Borland is described, with examples of its utilization. This class library is basic to almost any programming project and should be studied with the same attention as Borland C++ itself. The reuse of classes is one of features that makes C++ such a productive language. The chapter shows not only how to use the container classes directly but

also how to use them as base classes for your own customized classes. Both the `Object`-based and template-based container classes are described in detail.

Chapter 8, "Classes for Windows Programming," introduces Microsoft Windows from the viewpoint of a C++ programmer. It assumes familiarity with Windows as a graphical user interface and as a programming environment. The chapter deals with the design and development of classes to be used in a Windows environment. Each class is demonstrated with concise application programs, the code of which is available on the companion disk. The bulk of the literature available about Windows programming describes techniques that are excellent for C programmers but not sufficiently object-oriented. With Borland C++, classes are used to simplify the work and hide many of the usual difficulties of Windows programming.

Chapter 9, "A Complete Windows Program," combines all the knowledge gained from the preceding chapters to design and code a small Windows application program. Many of the Windows constructs are illustrated, including message boxes, dialog boxes, clipboard interfacing, using the printer, loading DLLs, and more. Using objects extensively can make tasks relatively easy and perhaps even enjoyable.

Chapter 10, "ObjectWindows Library Classes," explores the Borland Application Frameworks, starting with the ObjectWindows Library (OWL). The chapter describes ways to reuse the basic OWL classes to customize various parts of a typical Windows program. The approach taken in this chapter is *low level*, in that the focus is on single classes or Window objects rather than applications. Custom controls, persistent objects, splash images, and glyphs are among the topics covered. A basic understanding of OWL is required to follow the material in this chapter and Chapter 11.

Chapter 11, "OWL Applications," takes a higher-level approach to OWL Windows programming than Chapter 10. New classes are derived from OWL classes to support several common application requirements, such as status lines, pop-up menus, tool palettes, and edit windows. From a Windows programming perspective, Chapters 10 and 11 probably contain the most interesting material of the book.

Chapter 12, "Turbo Vision Classes," tackles another Borland application framework, Turbo Vision (TV). Many low-level classes are derived from built-in TV classes to customize the basic parts of a TV application, such as status lines, menu bars and desktops. Persistent TV objects are covered in detail at the end of the chapter. A basic understanding of TV is required to follow the material in this chapter and Chapter 13.

Chapter 13, "Turbo Vision Applications," uses the classes and examples developed in Chapter 12 to build several different TV applications. The applications emphasize some of the important features of TV, including context-sensitive help, property inspection, and edit windows. Using the guidelines shown in this chapter, you can develop sophisticated applications that incorporate a mouse, customizable colors, multiple overlapping windows, hot keys, and an integrated help facility. The chapter shows how to change the basic features of TV by deriving classes to suit your needs—without worrying about all the underlying details of event management, graphics modes, and so on.

Notational Conventions

A few basic conventions have been adopted throughout the book to increase readability:

1. When Borland C++ keywords are used in a sentence, they are printed in a special `monospace` type. This increases the clarity of the text, as in the following example:

   "When returning a `void` from a function..."

2. File names are printed in *italics*, as in the following example:

   "The definitions in *stdio.h* are used..."

3. Function names are printed in `monospace` and end with parentheses. When a file accepts parameters, three dots are used inside the parentheses to denote generic parameters:

   "The arguments of `printf(...)` are unknown at compile time..."

4. Variable names are printed in monospace:

   "Assigning a value to variable `arg` is allowable if..."

Requirements

You don't need a computer to study programming, but it sure helps! To master the material in this book, you not only need to study the source code of the various examples, but you should try making changes and compiling on your own. You need the following items:

- An IBM PC AT or compatible computer
- MS-DOS or PC-DOS 3.1 or later
- A Microsoft-compatible mouse
- EGA or VGA graphics
- Borland C++ version 3.0
- Borland Application Frameworks (for Chapters 10 through 13)
- Windows 3.0 or better (for Chapters 8 through 11)

The Microsoft Software Development Kit (SDK) for Windows is not required. If you have Turbo C++ or Borland C++ 2.0, you can still compile most of the code in chapters 1 through 6, but you won't be able to try the container examples in Chapter 7, the Windows code in Chapters 8 and 9, or the application frameworks in Chapters 10 through 13.

Contents

3 Inheritance 129