The APPRENTICE C++ PROGRAMMER

A Touch of Class

P. A. LEE

C. PHILLIPS

The Apprentice C++ Programmer: A Touch of Class

P. A. LEE and C. PHILLIPS

PWS Publishing Company

INTERNATIONAL THOMSON COMPUTER PRESS
I(T)P An International Thomson Publishing Company

London • Bonn • Johannesburg • Madrid • Melbourne • Mexico City • New York • Paris • Singapore
Tokyo • Toronto • Albany, NY • Belmont, CA • Boston, MA • Cincinnati, OH • Detroit, MI

The Apprentice C++ Programmer

I(T)P A division of International Thomson Publishing Inc.
The ITP logo is a trademark under licence

For more information, contact:

International Thomson Computer Press
Berkshire House
168–173 High Holborn
London WC1V 7AA
UK

PWS Publishing Company
20 Park Plaza
Boston
Massachusetts 02116-4324

http://www.itcpmedia.com

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

First printed 1996
Commissioning Editor: Samantha Whittaker
Typeset by Florencetype Ltd, Stoodleigh, Devon
Printed in Spain

ISBN (ITP edition) 1–85032–160–4
ISBN (PWS edition) 0–53495–339–5

The Apprentice C++ Programmer: A Touch of Class

Preface

Is this book for you?

Your initial reaction on picking up this book may well have been

"What, yet another book on C++? Why should I look at this one?"

If, however, you're looking for a fresh approach to C++ that:

- introduces object-oriented design and programming;
- assumes no prior knowledge of *any* programming language, nor of any mathematics beyond high-school level;
- emphasizes the importance of top-down design and object-oriented programming from the outset;
- integrates software engineering concerns;
- adopts an informal and gentle style that relies heavily on the presentation of examples of new topics based on real-world analogies before diving into a more formal description;

then read on, because this book is just what you've been searching for.

Unlike many other books on C++, you are *not* expected to know a fair amount of C and/or accept a bottom-up approach to C++ programming. If that's what you're looking for, then read no further; this book's not for you.

The target audience for this book is first-year undergraduates, or any others requiring an introduction to programming and object-oriented techniques. As such, this book closely follows the CS1 syllabus. The book is especially suitable for those who have never been exposed to programming before, as well as those who have some experience in other programming languages. Additionally, we

expect the material to be useful for those who have programming experience but require a gentle introduction to object-oriented techniques and C++.

Instructors contemplating adopting this book are encouraged to take a look at the supplementary materials available, particularly the Instructor's Manual, details of which are given later in this Preface.

AIMS AND OBJECTIVES

Our objective for people who follow this book, or courses based on it, is that they become *Apprentice C++ Programmers*. The parallel with a traditional engineering apprenticeship is apt. An apprentice programmer is someone who knows the basics of the programming trade, has a "toolbag" of useful paradigms and algorithms, and understands the tools of their trade. The apprentice programmer must also be able to apply programming skills to the problem in hand, and hence needs to acquire essential software engineering skills so that they can design, implement and maintain solutions to complex problems. Much more experience is needed to turn the apprentice into a practicing programmer, of course, but that's outside the scope of this book.

In summary, having completed an apprenticeship by following this book you will know:

- how to design and implement software systems in C++;
- the importance of object-oriented design and implementation;
- the importance of good software engineering practices.

You will also understand the relationship between these issues.

APPROACH

The fundamental approach in this book is the use of **Types** and **Objects**, and the use of these concepts pervades the whole process, from initial problem comprehension and solving, through to solution design and implementation. However, it's not feasible, nor desirable, in an apprenticeship to cover the whole spectrum of object-oriented analysis and design, especially that used in large (multi-member) software projects. Instead, we concentrate on the fundamentals: getting you to understand the concepts of objects and types, and how these can be used in algorithms to solve problems.

We don't believe that an introductory programming text (or course) can, or indeed should, cover all of the complexity inherent in object-oriented programming. Concepts like objects and types are hard enough to grasp without worrying

about polymorphism, multiple inheritance and the like. So, we introduce simple objects and types based on what used to be called Abstract Data Types (ADTs). Since we extend the idea of ADTs using the fundamental concepts of objects and types (encapsulation, information hiding, separation of interface and implementation), we introduce the terminology **Programmer Defined Types** (PDTs). The PDT concept is a useful stepping stone to full object-oriented techniques, which we leave to a more advanced text.

While the focus of the book is on teaching how to program, issues concerning "software engineering in-the-small" are also addressed and integrated into the explanations concerning programming. Many chapters contain sections addressing software engineering issues, so you can build up an appreciation of the engineering points at the same time as learning the programming language constructs.

Graphical notations are used extensively to support comprehension of the software engineering process:

- to represent the interface required by a user of a type;
- to represent the top-down design of algorithms, and the flow of control in a program (structured flowcharts);
- to show C++ syntax rules (syntax diagrams).

Projects and examples are used extensively throughout to provide exemplification of the programming and software engineering concepts. Rather than develop program components from scratch, the initial emphasis is on reuse of previously-developed components wherever possible, such as PDTs and functions, provided on an accompanying disk. Once you are familiar with using components, you'll be happier with implementing your own components.

The programming language C++ is used as the vehicle for introducing programming concepts because C++ has very good support for introducing new types. However, this book is *not* intended to be a thorough C++ reference text. No attempt is made to introduce the complete C++ syntax or semantics. Indeed, in many situations we introduce syntax to represent simplified C++ constructs that are only a sub-set of the full-blown language. There are several aspects of C++ that we positively discourage use of (particularly error-prone features of C++ inherited from C), and hence these are not mentioned at all. We introduce several new C++ types to provide better abstractions than those provided in the base language (for example, for arrays and character strings). A C or C++ hacker will probably be very disappointed in, or critical of, our usage of C++. We would regard this as highly satisfactory!

While we use C++ as the primary programming vehicle, concepts are introduced in three stages:

1. The concept in general, real-world terms.
2. The concept in general programming language terms.
3. The concept in C++ terms.

Our experience with teaching programming languages has been that those who have never programmed before have difficulty relating to programming concepts; hence the need for motivation in terms of real-world equivalents which you can relate to directly.

Additional features in the book include:

- a statement of objectives at the beginning of each chapter;
- a detailed summary at the end of each chapter indicating the skills learned;
- highlighted **lessons** throughout that summarize the key issues;
- highlighted **style rules** that promote the generation of well-engineered programs;
- numerous examples illustrating the new topics being introduced;
- end-of-section exercises with worked solutions providing a ready test of your understanding of the material covered;
- end-of-chapter exercises providing a more intensive examination.

OVERVIEW

The book is organized into three parts:

Part 1 **Concepts For Apprentice Programmers** covers the basic concepts of software engineering, problem solving and program design in an object-oriented style, and computers and their programming languages.

Part 2 **Learning To Use Your C++ Toolbag** introduces the programming language C++ and basic object-oriented features.

Part 3 **Practicing Your Trade** provides worked-through software projects.

Part 1

Part 1 sets the scene by giving an overview of software systems. This overview is more extensive than that traditionally found in an introductory programming text, for two important reasons. First, we assume no prior knowledge of computers or programming. Second, we wouldn't expect a software engineer immediately to sit down in front of a computer and start typing when presented

with a new problem. Instead, we'd expect them to analyze and understand the problem first to gain a clear idea of the required outcome. Similarly, we regard an understanding of what an apprenticeship in C++ programming involves as being essential before embarking on the apprenticeship itself. There's an apt analogy with constructing a building. Yes, you need to know about bricks and mortar, but you need a clear view of what the outcome is before you set about designing and building that palace or mud hut. So Part 1 concentrates on problem design and solving using types, permitting the design of object-oriented software, even though no details of a programming language have yet been presented.

Chapter 1 presents basic background concepts for people who don't really know what a computer or a program is, and tries to demystify the idea of a program as being something that only computer science specialists can understand. Chapter 2 introduces the basic ideas of problem solving for object-oriented software implementation – basic top-down ideas to start with, followed by simple object-oriented analysis techniques for producing designs for types and algorithms. By the end of Chapter 2 you should be able to produce a design for a solution to a problem, even though you won't yet know any C++ details. Chapter 3 introduces the fundamental features of a programming language, to provide background for the detailed C++ explanations that follow in Part 2.

Part 2

PDTs are used throughout Part 2 to provide a useful and interesting set of types from which interesting programs can be constructed very quickly. In the early chapters the emphasis is on *using* PDTs by means of their defined public interface without having to address the complexity of their implementation. For instance, a PDT representing a CD player is used extensively – we provide an implementation of this PDT that simulates the CD player's behavior graphically on a computer screen. You can write programs controlling an instance of a CD player and learn about declarations, repetition statements, and so on, with the graphical representation acting as a useful debugging aid. Later chapters address the issues of actually identifying, designing and implementing PDTs.

Chapters 4 through 15 introduce the C++ language details, with each chapter devoted to a single concept. The early emphasis is on changing existing programs so that interesting programming can be started early, rather than having to cover loads of syntax details before a first program can be run. We want to encourage you to concentrate on the programming details without spending a lot of time being confused by what the problem is you are trying to solve.

Part 3

Part 3 brings all the earlier issues of design, implementation and software engineering together in the form of worked-through large-scale software projects. Starting with an initial problem specification, the design and implementation phases are covered in detail. The final products are programs that illustrate almost all the features of C++ covered in the context of smaller problems in earlier chapters.

Chapter 16 puts into practice all the material covered in earlier chapters by tackling two major projects. Here the emphasis echoes that of Part 1, returning to those ideas of object-oriented analysis techniques for producing designs. Having reached this point you will be completely familiar with those programming details of Part 2, and hence able to complete the picture of the design and implementation of software products.

STUDENTS' READING PLAN

One of the key issues we address is the need to think carefully about the problem to be solved before attempting a solution. Too often the words "Write a program to . . ." are viewed as an instruction to go straight to a computer and start typing in some code. The result is a hacked piece of code that more often than not, bears little relationship with a well-engineered software product. Thus, while you may have a strong urge to dive straight into the C++ programming details of Chapter 4 onwards, we strongly recommend that you first cover the material in Chapters 1–3, taking a few weeks, during which time familiarity can be gained with a computing environment, so that you are at ease with this before getting thrown into the C++ details. Our experience is that those who have never programmed before are wary of these funny things called computers, and appreciate the background that the material of Part 1 gives them. The work involved in learning about the computer system, its interface, the editor interface, and the compilation system, as well as (possibly) a word-processing package, an e-mail tool, the World Wide Web, and so on, places you on a steep learning curve, when in fact the programming language details are complex enough to grasp by themselves!

Chapters 1 and 3 can be covered as quickly as necessary, depending on your experience. Whatever your background, you should take some time over Chapter 2, which addresses the issue of constructing a design before any implementation is started and shows the basic problem-solving strategies that a software engineer can adopt. The design emphasis is on identifying types and

objects, and trying to make you comfortable with the fundamental issues of types and objects (and their manipulation) before the C++ syntax is reached. The basic ideas of control-flow and sequencing are also introduced, without which it is difficult to understand programming at all. You then only need to concentrate on the C++ syntax details in Chapter 4 onwards, and the underlying concepts will be understood.

Having taken your time with Part 1, you'll be ready for the C++ programming details of Parts 2 and 3. Your progression thereon will for the most part be linear. However, Chapter 5 is largely reference material that can be skimmed on a first reading and returned to as programming themes in later chapters are developed.

PEDAGOGICAL DEVICES

Within the text we use various pedagogical devices. These devices, or text styles, allow us to differentiate the following things. First, sometimes we want to exemplify some concept that has just been described – a real-world analogy or an example to help you to understand that concept, and this is illustrated as follows.

EXAMPLE:

This is an example of the "Example" style device.

Sometimes we use the same device to introduce a fragment of C++ code that is illustrating a point that has just been made.

EXAMPLE:

```
// This is a valid C++ comment.
```

When we've been making an important point that you should take special care to note, we highlight such points in the following **lesson** style:

Lesson: This is an important lesson to learn.

For many C++ features we indicate some important style rules that should be followed when writing a program using that particular feature. In the text we highlight the important good and bad style points as follows.

✔ *This is a good style, that should be adopted.*
✗ *Whereas this bullet introduces a bad style that should be avoided.*

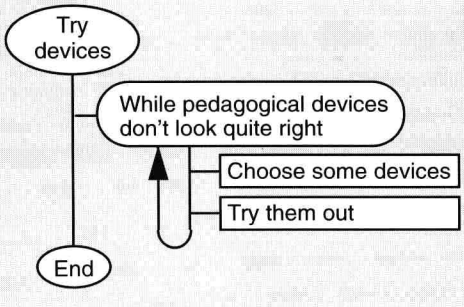
At other points in a chapter we wish to pose a problem for which a solution is to be developed during the chapter. The problem will often turn into a worked example with some or all of the following sections: problem, design, implementation, and output. We indicate these sections as follows.

PROBLEM:

Think about how to make a problem statement stand out in the text, so that it can be referred back to subsequently.

DESIGN:

Sometimes the design is explained in words, and often there is a design diagram.



IMPLEMENTATION:

```
Book thisBook;

// C++ code fragments
for (int chapter = 1; chapter <= 16; chapter += 1)
{
  thisBook.read(chapter);
}
```

OUTPUT:

This section shows examples of output produced. It may be textual output only, and hence shown in `courier` font. Sometimes screen shots are shown captured from an Apple Macintosh computer. The fact that they're from a Macintosh isn't important, and a particular system may differ in some of the details shown on the screen. All we want to do here is establish the link between a program and the output produced.

The other pedagogical devices we use in the text are icons in the margin. These icons are used to highlight different features.

For example, what do you think the icon on the left represents?

This icon is used to represent "stop-and-think". This is a point in the text where we want to pose a question whose answer is based on some material just presented. As here, the answer is given in the paragraph immediately following the paragraph labelled with the icon. But you should try to answer the question before reading on. It wouldn't be honest to do otherwise, would it? :-)

If you're asking "What are those funny characters on the end of that last paragraph?" then you've not seen a *smiley* before. (Turn the page clockwise by 90 degrees.) Smileys originated for computer-based text messages, to indicate when something was being stated humorously or not to be taken too seriously, and we've used them in this book for the same purpose.

We additionally use the following three icons:

The first icon is indicating an area in the C++ programming language that may change when a new standard version of C++ is agreed upon.

The floppy disk icon is indicating that a program being discussed is available in electronic form, and the words on the icon give the names of the appropriate directories or folders. As different computer systems have different conventions for naming their computer files, it may be necessary to search around on a given system to find the actual files (assuming they've been installed).

Finally, the skull-and-crossbones icon is used when there's a C++ feature that should be used with extreme care.

SUPPLEMENTARY MATERIALS

The complete teaching package for the *Apprentice C++ Programmer* consists of four main elements:

- this book itself, written to introduce programming in C++ in a system-independent format;
- a PC-compatible disk containing example programs and implementations of the PDTs employed in the book. We also provide implementations for three major platforms: Borland on the PC, Symantec on the Apple Macintosh, and cfront and g++ on UNIX. All software is available from our WWW server;
- a separately-available system-independent Lab Manual that can form the basis for practical classes, or can provide students with additional tutorial and practical exercises to try out newly-acquired skills. System-specific issues (such as descriptions of how to use the Symantec compiler on an Apple Macintosh, or Borland on a PC) are dealt with in documents available from our WWW server;
- an electronic Instructor's Pack containing all necessary material for the succesful delivery of a course based on this book, including a Manual that explains the background to the book material, and an extensive

set of slides in a variety of formats. WWW access is password-protected. Please contact ITP at:

```
http://www.itcpmedia.com/WebExtra/default.html
```

The Newcastle WWW address for on-line materials is:

```
http://www.cs.ncl.ac.uk/publications/books/apprenticeship/
```

ACKNOWLEDGEMENTS

It is a pleasure to acknowledge the help of many of our colleagues at Newcastle in the development of this text and its supporting material. Martin McLauchlan is largely responsible for producing the Lab Manual. Robert Stroud produced many of the basic classes that we make extensive use of, and has provided much useful feedback on the overall material. Linday Marshall was able to bail us out on some of the more obscure features of C++. Nigel Hall, Gerry Tomlinson, Jim Wight and Chris Ritson have undertaken sterling work in producing the support for the graphics on Macintosh, Unix and PC platforms. Thanks too to the hundreds of past and current students at Newcastle who have class-tested the material, and to all those not explicitly mentioned above who have provided support in the delivery of those classes.

The production of a book of this nature is a major task. The authors put the material together, but there is much work that goes on behind the scenes by the publishing staff before a manuscript comes to the market. Our Commissioning Editor, Samantha Whittaker, is worthy of special praise, not least for having to put up with authors who never failed to get in a dig at every opportunity. Thanks Sam, it's been a real pleasure working with you. We should also mention Mike Sugarman, John Normansell and Penny Grose for their encouragement and help throughout the development of this project.

Last, but by no means least, to our families. Only other authors know the burden they've had to bear! We solemnly promise not to get involved with any other book writing – well, not for a few weeks, at least :-)!

SOME FINAL OBSERVATIONS

Learning can be, and should be, a fun and enjoyable experience. Learning to become an apprentice software engineer is no different, so we hope students will enjoy the CD player simulation, drawing pretty patterns on a canvas,

discovering what the software inside a supermarket checkout register might look like, and maybe even some of our jokes. :-) Most of all, we hope students will be sufficiently encouraged by this apprenticeship to want to expand their knowledge of object-oriented programming in C++ still further.

Pete Lee, Chris Phillips
Department of Computing Science,
University of Newcastle upon Tyne,
UK
August 1996

P.A.Lee@newcastle.ac.uk
http://www.cs.ncl.ac.uk/people/p.a.lee/

Chris.Phillips@newcastle.ac.uk
http://www.cs.ncl.ac.uk/people/chris.phillips/