PREFACE

Welcome to the third edition of my C++ text. The highly successful first edition was one of the first textbooks available for teaching C++ in the first programming course. The text was introduced at the 1994 ACM Conference in Phoenix when many were arguing the virtues of teaching C++ and OOP versus Pascal and structured programming in the first programming course. I argued at that time, and still argue, that students need to be taught problem solving early-on using both the structured and object-oriented paradigms and, because of its hybrid nature, C++ is the only language suited to learning both of these paradigms. Since then, many institutions have made the switch from Pascal to C++ for just this reason, as well as the intense industry support for the C++ language. As a result, this third edition continues to provide an introduction to both structured and object-oriented problem solving techniques using the C++ language. Of course, many improvements have been made based on using the text in numerous classrooms all over the world since 1994.

As with earlier editions, the text starts from the beginning, assuming no previous knowledge of C, or any other programming language. The text is appropriate for any introductory programming (CS1) course using the C++ language as well as experienced programmers wanting an introduction to structured and object-oriented problem solving techniques using the C++ language.

New features in this third editions include:

- An introductory "getting acquainted" chapter that provides an overview of basic hardware and software concepts and gets students started early with the proverbial "Hello World" program.
- A comprehensive Visual C++ tutorial in Appendix C using Microsoft's popular Visual Studio integrated development environment (IDE).
- Learning modules strategically placed throughout the text on topics of GUIs, graphics, and software engineering. These modules are optional, but encouraged to provide a rich background for further study of these important topics.
- The use of the Unified Modeling Language (UML) class diagrams for object-oriented design.

- A series of software engineering modules that follow the Foo.com company programming team from a disastrous build-it, fix-it spaghetti code program, to a structured design, and finally an object-oriented design of their program using the Unified Modeling Language, UML. With these modules, the student gets an understanding of the bigger software engineering picture. In addition, the UML module prepares the student for planning and designing their own classes in Chapter 10.
- The use of the ANSI/ISO *string* class throughout. However, C-strings are also covered in the chapter on arrays.
- Coverage of the ANSI/ISO Standard Template Library (STL) *vector, stack, queue,* and *list* classes.
- A text CD that includes an introductory edition of Microsoft's Visual C++ as well as all example, case study, and Visual C++ tutorial source code.
- An instructor's CD that includes PowerPoint presentations organized by chapter, a test bank with answer keys which includes chapter tests as well as comprehensive mid-term and final exams, the solutions to all the chapter questions and programming problems, and all the example, case study and tutorial source code.
- A companion Web site at *www.prenhall.com/staugaard* that includes chapter objectives, on-line review tests that are automatically graded, a glossary of terms, downloadable example, case study and tutorial source code, and a syllabus manager.

APPROACH

The text emphasizes problem solving techniques using the C++ language. In fact, problem solving is the essential theme throughout the text. The student begins mastering the art of problem solving in Chapter 2, using problem abstraction and stepwise refinement via the "Programmer's Algorithm." Emphasis is first based on the structured (procedural) paradigm building *gently* into the object-oriented paradigm using seeds planted in the earlier chapters. This approach gradually prepares the student for in-depth coverage of classes and objects later in the text, while building essential structured programming concepts.

This third edition is based on the highly successful earlier editions that have been widely used in CS1 courses since 1994. Previous editions have been adopted at large and small institutions alike. One of the many reasons for the success of these earlier editions is that the text is highly readable and student oriented, with a teachable pedagogy and excellent features. The text provides sufficient material for a fast-paced one semester course or slower paced two semester course sequence.

FEATURES

- Highly readable and student oriented.
- Thoroughly instructor and student tested via first and second edition adoptions at large and small institutions alike.
- Early introduction to problem abstraction and stepwise refinement in Chapter 2 and used as a theme throughout the text using "Problem Solving In Action" case studies.
- A *gentle* introduction to classes and objects, beginning in Chapter 2. In-depth coverage of C++ classes and objects in Chapter 10.

- ❒ The "Programmer's Algorithm": A step-by-step process used to get students started on the right programming tract by considering problem definition, solution planning via algorithms, and good documentation. Problem solving using problem abstraction and stepwise refinement, prior to coding, is stressed throughout the text. All "Problem Solving in Action" case studies follow these proven software engineering techniques.
- ❒ Tip and note boxes for students throughout.
- ❒ Section-by-section quick-check exercises for students to check their progress with all answers in an appendix.
- ❒ Examples that pose problems and then give solutions immediately.
- ❒ Chapter questions with solutions on an instructor's CD.
- ❒ Chapter programming problems with solutions on an instructor's CD.
- ❒ Comprehensive glossary of general computer science as well as object-oriented programming terms.

TO THE INSTRUCTOR

This text has been written to teach structured and object-oriented problem solving techniques using the C++ language at the freshman level in a CS1 type of course. In today's market, it is imperative that students know both paradigms. Students need to understand the roles of and relationship between classes and objects early-on while at the same time learning classic program structuring techniques. I have found that there is no "paradigm shift" when class and object concepts are integrated into the structured paradigm. Structured, or procedural, programming is built around functions, and object-oriented programming is built around classes. Do the two have any relationship whatsoever? Yes! The classes that we build are constructed using elements of structured programming, namely functions. As a result, the structured paradigm is embedded within the object-oriented paradigm. This is why we need to study structured programming first, integrating object-oriented concepts where the opportunity arises, and *gently* move into object-oriented programming.

Some will say that you can't teach programming using C++, because the language is too complicated. I disagree. There is no need to teach every detail of the language in a beginning course. I have used a subset of C++ to teach fundamental structured and object-oriented concepts and have found that beginning students do not have any more difficulty using C++ than Pascal or JAVA with this approach. In addition, learning C++ has the added benefit for the student of learning a very widely used industry standard.

The text can be taught in one or two terms, depending on the ability of the students. In a two term sequence, I would suggest coverage through the topic of functions (Chapters 1–8). Then, begin the second term with arrays and finish out the book (Chapters 9–15). Make sure to cover the strategically placed learning modules on software engineering. These modules provide the student with an understanding of the bigger software engineering picture.

The text begins with a "getting acquainted" chapter that provides an overview of fundamental hardware/software concepts and ends with a section on getting started with C++. If you are using Visual C++, I suggest that you assign the first part of the

Visual C++ tutorial in Appendix C at this time so that students learn how to enter, compile, and execute a C++ program in the Visual C++ environment. Chapter 2 is devoted entirely to problem solving using problem abstraction and stepwise refinement. These concepts are presented in detail here and used as a theme throughout the text. The chapter discusses problem solving using what I call "The Programmer's Algorithm" and should be covered thoroughly. The programmer's algorithm is a step-by-step process that I have used to get students started on the right programming track by considering problem definition, step-by-step solution planning via algorithms, and good documentation. I have employed a pseudocode algorithmic language for problem solution that is generic, simple, and allows for easy translation to the coded C++ program.

Chapter 3 introduces the concepts of data abstraction and ADTs as well as traditional data types, classes, and objects. Here is where we begin planting the seeds of object-orientation to prevent any possible "paradigm shift" for the student when object-oriented programming is covered in-depth in Chapter 10. I suggest that you emphasize the concepts of classes, objects, and ADTs here to get students accustomed to object-oriented concepts. In addition, Chapter 3 introduces students to C++ functions, the common denominator of both paradigms.

In Chapters 4–7, students learn about program I/O, decision making, and iteration. These chapters provide the "nuts and bolts" required to write workable C++ programs. The sequence, decision, and iteration control structures available in C++ are covered thoroughly, emphasizing program logic and the required C++ syntax. The common pitfalls of program logic design are pointed out throughout this material via program debugging tips and programming notes. Again, I have integrated object-oriented concepts within these traditional structured programming topics, especially in Chapter 4 when using the C++ I/O classes and objects. Chapter 4 also introduces files so that students can begin reading and writing their own files early. Again, this opportunity is used to integrate object-oriented concepts into the discussion through the use of C++ file stream classes and objects. Detailed file manipulation is covered later, in Chapter 12.

At this point, your students have a solid knowledge of the important role of functions for both the structured and object-oriented paradigms. Since Chapter 2, they have been designing structured programs using functional decomposition and have observed the role of functions relative to object behavior. In Chapter 8, functions are covered in-depth. This chapter has been greatly improved over time, based on my experience of teaching C++ functions. I have incorporated easy-to-follow guidelines for building function interfaces relative to the topics of return values and parameter passing. It is critical that students understand this material in order to become successful C++ programmers and understand the object-oriented concepts presented in later chapters. The last section of this chapter discusses the important topic of problem solving with recursion. Recursion is introduced via a simple compound interest example, followed by and in-depth discussion of how recursion works and when it should be applied.

Students get their first exposure to data structures in Chapter 9, which discusses one-dimensional arrays. Again, I feel it is important that students have a solid knowledge of arrays and their manipulation. As result, this chapter covers traditional C-type arrays, including C-strings, and not some array class built by the au-

thor to hide the details and pitfalls associated with arrays. I have also used this topic to present several classic searching and sorting algorithms that are essential at this level. The ANSI/ISO *vector* class is discussed at the end of the chapter, after the students have mastered C-type arrays.

At this point, the student is prepared to learn about classes and objects in-depth. Again, there will be no paradigm shift here, because we have integrated these topics into the course since Chapter 2. In-depth coverage of classes and objects is provided in Chapter 10 and a solid introduction to class inheritance is provided in Chapter 11. In Chapter 10, students learn how to design their own classes using UML and convert their UML class diagrams into workable object-oriented programs using C++. Chapter 11 expands on the material in Chapter 10 by discussing inheritance. Many texts avoid the topic of inheritance. However, inheritance is one of the cornerstones of OOP and is relatively easy to cover at this point. Chapter 11 provides the student with a solid understanding of inheritance through a comprehensive banking example. The chapter closes with a discussion of polymorphism and dynamic binding.

File manipulation is discussed in-depth in Chapter 12. Here, the student learns how to read, write, append, and change disk files using C++. The topic of files in C++ provides a great opportunity to show how C++ employs inheritance to create reusable code. Here, the student learns how to employ inheritance through the C++ file stream class hierarchy to implement files.

The important topic of pointers is "addressed" in Chapter 13 in preparation for their use to build linked data structures in the next chapter. Pointers are fundamental to programming in C++ and should be covered thoroughly. This material, combined with the material on classes and objects in the previous two chapters, prepares the student to learn about the classic stack, queue, and linked-list ADTs in Chapter 14.

Chapter 14 provides an introduction to ADTs in preparation for an advanced course in data structures. The classic stack, queue, and linked list ADTs are covered thoroughly. In this chapter, ADTs are covered at two levels: The purely abstract level using the black box interface approach, and at the implementation level using C++ classes. Here is where the student really appreciates object-oriented programming, because ADTs are naturally implemented using C++ classes. The chapter closes with a discussion of the ANSI/ISO *stack, queue,* and *list* classes, after the students have mastered the building of their own stack, queue, and list classes using arrays and pointers.

The text closes with a chapter on multidimensional arrays. The chapter focuses primarily on the manipulation of two-dimensional arrays and closes with a comprehensive case study which applies two-dimensional arrays to simultaneous equation solution using Cramer's rule. If you wish, you can cover this chapter immediately after covering one-dimensional arrays in Chapter 9 without any loss in continuity.

The code in the text is portable and meets the new ANSI/ISO C++ standard as close as possible at this time. Finally, the following supplements are available to support this text:

- ❒ An instructor's CD that includes PowerPoint presentations organized by chapter, a test bank with answer keys that includes chapter tests as well as comprehensive mid-

term and final exams, the solutions to all the chapter questions and programming problems, and all the example, case study, and tutorial source code.

- A companion Web site at *www.prenhall.com/staugaard* that includes chapter objectives, on-line review tests that are automatically graded, a glossary of terms, downloadable example, case study and tutorial source code and a syllabus manager.
- A student laboratory manual and workbook for those using TURBO C++ that provides over 30 laboratory exercises that guide the student through program development with minimal instructor involvement. (ISBN 0-13-63962-5)

TO THE STUDENT

The market demands that computing professionals know the latest problem solving, programming, and software engineering techniques. This book has been written to provide you with an introduction to both structured and object-oriented programming techniques—two of the most important and widely used techniques within the industry. The text emphasizes problem solution through the use of problem abstraction and stepwise refinement throughout. You will learn to attack problems using a "top/down" as well as an "inside-out" strategy. By the end of the text you will have the knowledge required to solve complicated problems using either the top/down structured approach or inside-out object-oriented approach.

Make sure that you go through all the examples and "Problem Solving in Action" case studies as well as the software engineering learning modules. These have been written in short, understandable modules that stress the fundamental concepts being discussed. The case studies and learning modules are integrated into the text at key points in an effort to tie things together and prepare you for future topics.

Finally, above all, get your hands dirty! You cannot become a competent programmer by just reading this book and listening to your instructor's lectures. You must get your hands dirty at the machine by developing *your own* C++ programs. Get started early by sitting down at a computer, getting acquainted with your C++ compiler, and writing *your own* C++ programs. In fact, included on the CD that accompanies this text is Microsoft's introductory version of Visual C++. I suggest that you get your hands dirty right now and install this program on your system. Then, go through the first part of the Visual C++ tutorial in appendix C to learn how to enter, compile, and run a C++ program using Visual C++.

TO THE PROFESSIONAL

The C++ programming language has taken the commercial software development industry by storm. Most of today's window-based software is being developed using C++ because of its object-orientation. Whether you are an experienced programmer or a novice, you will find that this book provides the "nuts and bolts" required to get you writing C++ programs quickly. In fact, this book is all that you will need to begin learning the C++ language. The text provides comprehensive coverage of structured programming using the C++ language and a sound introduction to

object-oriented programming, both of which are essential in today's programming market. An introductory version of Microsoft's Visual C++ is provided on the CD that accompanies this book. In addition, a Visual C++ tutorial is provided in appendix C to get you started using Visual C++ quickly.

ACKNOWLEDGMENTS

Contributions to this text have come from many circles. From the academic world I would like to thank David Teague, Western Carolina University; Kathy Liszka, University of Akron; David Hudak, Ohio Northern University; Buster Dunsmore, Purdue University; George Luger, University of New Mexico; Keith Pierce, University of Minnesota; and Bob Holloway, University of Wisconsin, at Madison. All have reviewed the manuscript over the years and have made valuable contributions.

From the industrial world, I would like to thank Bjarne Stroustrup of Bell Labs, now part of Lucent Technologies, who reviewed the first edition manuscript and made many valuable suggestions relative to teaching C++ to beginning students as well as the language philosophy and details.

From the student world, I would like to thank my own students who, over that past ten years, have inspired the creation of this text and have made many valuable suggestions. Special thanks to one of my former students, Riley White, who prepared the Visual C++ tutorial material in Appendix C as well as the Windows template used in the GUI and graphics modules.

From the publishing world of Prentice Hall, I would like to thank my acquisitions editor, Petra Recter, the assistant editor, Sarah Burrows, and the production editor, Patty Donovan of Pine Tree Composition, Inc.

Enjoy!

Andrew C. Staugaard, Jr.