PREFACE

Problem Solving and Program Design in C teaches a disciplined approach to problem solving, applying widely accepted software engineering methods to design program solutions as cohesive, readable, reusable modules. We present as an implementation vehicle for these modules a subset of ANSI C—a standardized, industrial-strength programming language known for its power and portability. This text can be used for a first course in programming methods: It assumes no prior knowledge of computers or programming. The text's broad selection of case studies and exercises allows an instructor to design an introductory programming course in C for computer science majors or for students from a wide range of other disciplines.

New to this Edition

Several changes to this edition are listed below:

- Chapters 3 (Functions), 5 (Loops), and 7 (Arrays) include optional sections on graphics programming
- Chapter 6 (Pointers and Modular Programming) includes a new section 6.1 on pointers
- New complete programs show use of if statements in Chapter 4
- New complete program shows use of switch statement in Chapter 4
- Chapter 7 (Simple Data Types) in previous edition is eliminated and its contents integrated into other chapters of the book
- Hardware examples in Chapter 1 are updated to reflect current technology
- Several chapters contain new programming project homework problems

More About Graphics Many Computer Science faculty have recommended the use of graphics to help motivate the study of introductory programming and as a vehicle to help students understand how to use libraries and to call functions. We agree with this viewpoint and have included three optional sections with graphics examples in this edition. The new graphics sections include:

Section 3.6: Introduction to Computer Graphics
Section 5.11: Loops in Graphics Programs
Section 7.10: Graphics Programs with Arrays

To reduce the overhead required to introduce graphics, we decided to use WinBGIm (Windows BGI with mouse), which is a package based on the Turbo

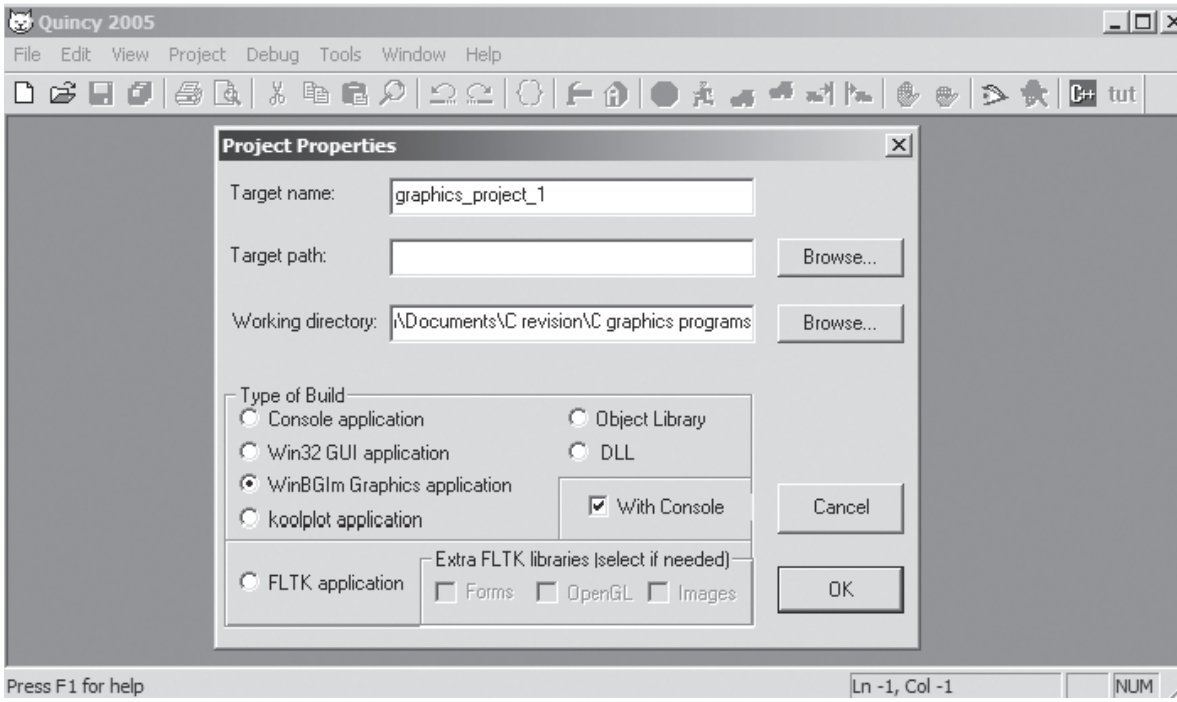

FIGURE 1

Pascal BGI (Borland Graphics Interface) library. WinBGIm was created to run on top of the Win32 library by Michael Main and his students at the University of Colorado. Several development platforms appropriate for CS 1 courses have incorporated WinBGIm. Quincy (developed by Al Stevens) is an open-source student-oriented C++ IDE that includes WinBGIm as well as more advanced libraries (http://www.codecutter.net/tools/quincy). Figure 1 shows the Quincy new project window (File → New → Project) with WinBGIm Graphics application selected.

A command-line platform based on the open-source GNU g++ compiler and the emacs program editor is distributed by the University of Colorado (http://www.codecutter.net/tools/winbgim). WinBGIm is also available for Bloodshed Software's Dev-C++ and Microsoft's Visual Studio C++.

Using C to Teach Program Development

Two of our goals—teaching program design and teaching C—may be seen by some as contradictory. C is widely perceived as a language to be tackled only after one has learned the fundamentals of programming in some other, friendlier language. The

perception that C is excessively difficult is traceable to the history of the language. Designed as a vehicle for programming the UNIX operating system, C found its original clientele among programmers who understood the complexities of the operating system and the underlying machine and who considered it natural to exploit this knowledge in their programs. Therefore, it is not surprising that many textbooks whose primary goal is to teach C expose the student to program examples requiring an understanding of machine concepts that are not in the syllabus of a standard introductory programming course.

In this text, we are able to teach both a rational approach to program development and an introduction to ANSI C because we have chosen the first goal as our primary one. One might fear that this choice would lead to a watered-down treatment of ANSI C. On the contrary, we find that the blended presentation of programming concepts and of the implementation of these concepts in C captures a focused picture of the power of ANSI C as a high-level programming language, a picture that is often blurred in texts whose foremost objective is the coverage of all of ANSI C. Even following this approach of giving program design precedence over discussion of C language features, we have arrived at coverage of the essential constructs of C that is quite comprehensive.

Pointers and the Organization of the Book

The order in which C language topics are presented is dictated by our view of the needs of the beginning programmer rather than by the structure of the C programming language. The reader may be surprised to discover that there is no chapter entitled "Pointers." This missing chapter title follows from our treatment of C as a high-level language, not from an absence of awareness of the critical role of pointers in C.

Whereas other high-level languages have separate language constructs for output parameters and arrays, C openly folds these concepts into its notion of a pointer, drastically increasing the complexity of learning the language. We simplify the learning process by discussing pointers from these separate perspectives where such topics normally arise when teaching other programming languages, thus, allowing a student to absorb the intricacies of pointer usage a little at a time. Our approach makes possible the presentation of fundamental concepts using traditional high-level language terminology—output parameter, array, array subscript, string—and makes it easier for students without prior assembly language background to master the many facets of pointer usage.

Therefore, this text has not one but four chapters that focus on pointers. Chapter 6 (Pointers and Modular Programming) begins with a discussion of pointers, indirect reference, and the use of pointers to files (moved from Chapter 2). It then discusses the use of pointers as simple output and input/output parameters, Chapter 7 deals with arrays, Chapter 8 presents strings and arrays of pointers. Chapter 11 discusses file pointers again. Chapter 13 describes dynamic memory allocation after reviewing pointer uses previously covered.

Software Engineering Concepts

The book presents many aspects of software engineering. Some are explicitly discussed and others are taught only by example. The connection between good problem-solving skills and effective software development is established early in Chapter 1 with a section that discusses the art and science of problem solving. The five-phase software development method presented in Chapter 1 is used to solve the first case study and is applied uniformly to case studies throughout the text. Major program style issues are highlighted in special displays, and the coding style used in examples is based on guidelines followed in segments of the C software industry. There are sections in several chapters that discuss algorithm tracing, program debugging, and testing.

Chapter 3 introduces procedural abstraction through selected C library functions, parameterless void functions, and functions that take input parameters and return a value. Chapters 4 and 5 include additional function examples including the use of a function as a parameter and Chapter 6 completes the study of functions that have simple parameters. The chapter discusses the use of pointers to represent output and input/output parameters.

Case studies and sample programs in Chapters 6, 7, and 10 introduce by example the concepts of data abstraction and encapsulation of a data type and operators. Chapter 12 presents C's facilities for formalizing procedural and data abstraction in personal libraries defined by separate header and implementation files. Chapter 14 (on the textbook website) introduces essential concepts of multiprocessing, such as parent and child processes, interprocess communication, mutual exclusion locking, and dead lock avoidance. Chapter 15 (on the textbook website) describes how object-oriented design is implemented by C++.

The use of visible function interfaces is emphasized throughout the text. We do not mention the possibility of using a global variable until Chapter 12, and then we carefully describe both the dangers and the value of global variable usage.

Pedagogical Features

We employ the following pedagogical features to enhance the usefulness of this book as a learning tool:

End-of-Section Exercises Most sections end with a number of Self-Check Exercises. These include exercises that require analysis of program fragments as well as short programming exercises. Answers to selected Self-Check Exercises appear online at www.aw.com/cssupport in the directory for "Hanly".

Examples and Case Studies The book contains a wide variety of programming examples. Whenever possible, examples contain complete programs or functions rather than incomplete program fragments. Each chapter contains one or more substantial case studies that are solved following the software development method. Numerous case studies give the student glimpses of important applications of

computing, including database searching, business applications such as billing and sales analysis, word processing, and environmental applications such as radiation level monitoring and water conservation.

Syntax Display Boxes The syntax displays describe the syntax and semantics of new C features and provide examples.

Program Style Displays The program style displays discuss major issues of good programming style.

Error Discussions and Chapter Review Each chapter concludes with a section that discusses common programming errors. The Chapter Review includes a table of new C constructs.

End-of-Chapter Exercises Quick-Check Exercises with answers follow each Chapter Review. There are also review exercises available in each chapter.

End-of-Chapter Projects Each chapter ends with Programming Projects giving students an opportunity to practice what they learned in the chapter.

Appendices

Reference tables of ANSI C constructs appear on the inside covers of the book. Because this text covers only a subset of ANSI C, the appendices play a vital role in increasing the value of the book as a reference. Throughout the book, array referencing is done with subscript notation; Appendix A is the only coverage of pointer arithmetic. Appendix B is an alphabetized table of ANSI C standard libraries. The table in Appendix C shows the precedence and associativity of all ANSI C operators; the operators not previously defined are explained in this appendix. Appendix D presents character set tables, and Appendix E lists all ANSI C reserved words.

Supplements

The following supplemental materials are available to all readers of this book at www.pearsonhighered.com/irc:

- Source code
- Known errata
- Answers to odd-numbered Self-Check exercises.

The following instructor supplement is available only to qualified instructors at the Pearson Instructor Resource Center. Visit www.pearsonhighered.com/irc or contact your local Pearson sales representative to gain access to the IRC.

- Solutions Manual

Acknowledgments

Many people participated in the development of this textbook. For this edition, we want to thank Michael Main for his assistance with WinBGIm and help with some of the graphics examples. We would also like to acknowledge the contributions of his students at the University of Colorado who adapted WinBGI to create WinBGIm (Grant Macklem, Gregory Schmelter, Alan Schmidt, and Ivan Stashak). The reviewers for this edition were Frank L. Friedman, Temple University, Philadelphia, PA; Mark S. Hutchenreuther, California Polytechnic State University, San Luis Obispo, CA; Anwar Mamat, University of Nebraska, Lincoln, NE; Hamdy Soliman, New Mexico Tech, Socorro, NM; Tami Sorgente, Florida Atlantic University, Boca Raton, FL; and Alexander Stoychev, Iowa State University, Ames, IA.

We also want to thank Charlotte Young of South Plains College for her help in creating Chapter 0, and Jeff Warsaw of WaveRules, LLC, who contributed substantially to Chapter 14. Joan C. Horvath of the Jet Propulsion Laboratory, California Institute of Technology, contributed several programming exercises, and Nelson Max of the University of California, Davis suggested numerous improvements to the text. Jeri appreciates the assistance of her Loyola College in Maryland colleagues—James R. Glenn, Dawn J. Lawrie, and Roberta E. Sabin—who contributed several programming projects. We are also grateful for the assistance over the years of several Temple University, University of Wyoming, and Howard University former students who helped to verify the programming examples and who provided answer keys for the host of exercises, including Mark Thoney, Lynne Doherty, Andrew Wrobel, Steve Babiak, Donna Chrupcala, Masoud Kermani, Thayne Routh, and Paul Onakoya.

It has been a pleasure to work with the Pearson team in this endeavor. The Editor-in-Chief, Michael Hirsch, along with the Senior Project Manager, Carole Snyder provided guidance and encouragement throughout all phases of manuscript revision. Pat Brown and Bob Engelhardt supervised the production of the book, while Yez Alayan developed the marketing campaign.

J.R.H.
E.B.K.