ABOUT THE AUTHOR

I first began teaching introductory computer science more than 20 years ago while I was still a student at Harvard. Since receiving my Ph.D. in 1980, I have taught computer science at Harvard, Wellesley, and Stanford, where I am Associate Chair of the Computer Science Department. In that capacity, I am responsible for the undergraduate program in computer science. Although I have taught advanced courses in computer science and have also worked in the research industry, my greatest joy comes from opening up the enormous power of computers to students who are just beginning to learn about them. In their excitement, my own love for computer science is constantly renewed.

In addition to my teaching at Stanford, I have served since 1990 as the president of Computer Professionals for Social Responsibility, a public-interest association of computer professionals with 2000 members in 22 chapters throughout the United States. Computers affect our society in many different ways. Just as it is important to learn about the technology, it is critical that we also take the responsibility to ensure that computers are used for the benefit of all. If you have suggestions as to how I might make the presentation more clear, or you encounter errors in this text, please let me know. You can reach me by electronic mail at ericr@aw.com.

Eric S. Roberts
Stanford University

TO THE STUDENT

Welcome! By picking up this book, you have taken a step into the world of computer science -- a field of study that has grown from almost nothing half a century ago to become one of the most vibrant and active disciplines of our time.

Over that time, the computer has opened up extraordinary possibilities in almost every area of human endeavor. Business leaders today are able to manage global enterprises on an unprecedented scale because computers enable them to transfer information anywhere in a fraction of a second. Scientists can now solve problems that were beyond their reach until the computer made the necessary calculations possible. Filmmakers use computers to generate dramatic visual effects that are impossible to achieve without them. Doctors can determine much more accurately what is going on inside a patient because computers have enabled a massive transformation in medical technology.

Computers are a profoundly empowering technology. The advances we have seen up to now are small compared to what we will see in the next century. Computers will play a major role in shaping that century, just as they have the last 50 years. Those of you who are students today will soon inherit the responsibility of guiding that progress. As you do so, knowing how to use computers can only help.

Like most skills that are worth knowing, learning how computers work and how to control their enormous power takes time. You will not understand it all at once. You must start somewhere. Twenty-five centuries ago, the Chinese philosopher Lao-tzu observed that the longest journey begins with a single step. This book can be your beginning.

For many of you, however, the first step can be the hardest to take.

Many students find computers overwhelming and imagine that computer science is beyond their reach. Learning the basics of programming, however, does not require advanced mathematics or a detailed understanding of electronics. What matters in programming is whether you can progress from the statement of a problem to its solution. To do so, you must be able to think logically. You must have the necessary discipline to express your logic in a form that the computer can understand. Perhaps most importantly, you must be able to see the task through to its completion without getting discouraged by difficulties and setbacks. If you stick with the process, you will discover that reaching the solution is so exhilarating that it more than makes up for any frustrations you encounter along the way.

This book is designed to teach you the fundamentals of programming and the basics of C, which is the dominant programming language in the computing industry today. It treats the whys of programming as well as the hows, to give you have a feel for the programming process as a whole. It also includes several features that will help you focus on the essential points and avoid errors that slow you down. The next few pages summarize these features and explain how to use this book effectively as you begin your journey into the exciting world of computer science.

TO THE INSTRUCTOR

In 1991-92, Stanford decided to restructure its introductory computer science curriculum to use ANSI C instead of Pascal. We chose to adopt ANSI C in our introductory courses for the following reasons:

- Students demanded a more practical language. Future employers want students to have more direct experience with the tools industry uses, which today are principally C and C++. With few employment opportunities listed for Pascal programmers in the newspaper employment section, our students began to question the relevance of their education.

- Our Pascal-based curriculum taught students to program in a language that they would never again use. We discovered that many of those students, when they abandoned Pascal for more modern languages, often forgot everything they had learned about programming style and the discipline of software engineering. Having now taught these skills in the context of a language that the students continue to use, we have found that they end up writing much better programs.

- Many of our advanced computer science courses, particularly in the systems area, require students to program in C. Working with C from the beginning gives students much more experience by the time they reach the advanced courses.

- Learning C early paves the way to learning C++ and the object-oriented paradigm. Because our students have a strong background in C programming after their first year of study, we have been able to offer our course on object-oriented system design much earlier in the curriculum.

- C makes it possible to cover several important topics, such as modular development and abstract data types, that are hard to teach in a Pascal environment.

- In the last five years, C has made significant headway toward replacing Fortran as the lingua franca of programming for the engineering sciences.

Given our experience over the last two years, I am convinced that the choice was a good one and that our program is stronger because of the change.

At the same time, it is important to recognize that teaching C in the first programming course is not always easy. C was designed for programmers, not introductory students. Many of its features make sense only when understood in terms of a larger conceptual framework that new students do not recognize. In many respects, C is a complex language. To teach it at the introductory level, we must find a way to control its complexity.

THE LIBRARY-BASED APPROACH

One of the central goals of this text is to enable teachers to manage C's inherent complexity. Managing complexity, however, is precisely what we do as program- mers. When we are faced with a problem that is too complex for immediate solution, we divide it into smaller pieces and consider each one independently. Moreover, when the complexity of one of those pieces crosses a certain threshold, it makes sense to isolate that complexity by defining a separate abstraction that has a simple interface. The interface protects clients from the underlying details of the abstraction, thereby simplifying the conceptual structure.

The same approach works for teaching programming. To make the material easier for students to learn, this text adopts a library-based approach that emphasizes the principle of abstraction. The essential character of that approach is reflected in the following two features that set this book apart from other introductory texts:

1. Libraries and modular development -- essential concepts in modern programming -- are covered in considerable detail early in the presentation. Part II focuses entirely on the topics of libraries, interfaces, abstractions, and modular development. Students learn how to use these techniques effectively before they learn about arrays.

2. The text demonstrates the power of libraries by using them. It is one thing to tell students that libraries make it possible to hide complexity. It is quite another to demonstrate that concept. This text introduces several new libraries that hide details from the students until they are ready to assimilate them. The libraries give students th e power to write useful programs that they could not develop on their own. Later chapters reveal the implementation of those libraries, thereby allowing students to appreciate the power of abstraction.

In 1992, I attempted to teach the introductory course using only the ANSI libraries. The results were not encouraging. Each new topic required that the student understand so much about the rest of C that there was no effective way to present the material. For example, students had to understand the mechanics of arrays, pointers, and allocation before they could use string data in any interesting way, even though string manipulation is simpler conceptually. My best

students managed to understand what was going on by the end of the quarter. Most, however, did not. Since we introduced the library-based approach in 1993, students have assimilated the material more easily and learned much more about computer science.

The library interfaces and associated implementations used in this text are reprinted in Appendix B, which also gives instructions for obtaining the source code electronically through anonymous FTP (File Transfer Protocol).

THE ORDER OF PRESENTATION

This book presents topics in the same order as Stanford's introductory course, except for the material in Chapter 17, which we cover in the second course. Depending on your audience and the goals of your course, you may want to vary the order of presentation. The following notes provide an overview of the chapters and indicate some of the more important dependencies.

Chapter 1 traces the history of computing and describes the programming process. The chapter requires no programming per se but provides the contextual background for the rest of the text.

I have designed Chapters 2 and 3 for students with little or no background in programming. These chapters are conceptual in their approach and focus on problem solving rather than on the details of the C language. When new students are faced with detailed rules of syntax and structure, they concentrate on learning the rules instead of the underlying concepts, which are vastly more important at this stage. If your students already know some programming, you could move much more quickly through this material.

Chapters 2 and 3 are informal in their approach and concentrate on developing the student's problem-solving skills. Along the way, they introduce several basic statement forms and control structures, but only as idioms to accomplish a specific task. Chapter 4 adds formal structure to this topic by describing each statement form in turn, detailing its syntax and semantics. The chapter also includes an extensive discussion of Boolean data.

Chapter 5 introduces functions and procedures. It begins with simple examples of functions and then continues with increasingly sophisticated examples. The mechanics of parameter passing are discussed in a separate section that includes many diagrams to help students follow the flow of data from one function to another. The chapter ends with a significant programming example that illustrates the concept of stepwise refinement.

The algorithmic concepts presented in Chapter 6 are fundamental to computer science but may not be required for all students. If your audience consists of engineers or potential computer science majors, the material will prove extremely valuable. For less technical audiences, however, you can eliminate much of this material without disturbing the flow of the presentation.

I have found that integrating graphics in the introductory course is a great way to increase student interest in the material. Chapter 7 exists for that reason. At this stage, students have learned the mechanics of functions but have no burning need to write them. Letting students draw complex pictures on the screen gives them that incentive.

The graphics library is implemented for several of the major programming environments and can therefore be used in most institutions.

Chapter 8 has two themes, which are in some sense separable. The first part of the chapter discusses design criteria for interfaces and is essential for anyone who needs to understand how modern programming works. The second part of the chapter applies those principles to build a random number library. The random.h interface itself is less important than the general principles, although use of the random number library is required for many exercises throughout the text.

Chapter 9 introduces strings as an abstract type and represents, to a certain extent, the cornerstone of the library-based approach. By using a dynamic string library, students can easily write programs that perform sophisticated string manipulation, even though they do not yet understand the underlying representation, which is covered in Chapter 14. Introducing strings at this point in the presentation enables students to write much more exciting programs that they could otherwise.

On a first reading, it is easy to miss the purpose of Chapter 10, which appears to be an extension of the discussion of strings begun in Chapter 9. The fundamental value of Chapter 10 does not lie in the Pig Latin program, which is more fun than it is practical. The entire reason for the example is that it provides the motivation to build the scanner interface used to separate words on the input line. The scanner module proves its usefulness over and over again, not only in the first course, but in subsequent courses as well. It is the most practical tool students create in the course and therefore serves as a compelling example of the value of modularity.

Chapters 11 through 16 introduce the fundamental compound structures -- arrays, records, pointers, and files -- in an order that has worked well in practice. Because the base language is C, it is important to present pointers as soon as possible after introducing arrays so that you can emphasize the connections between them. Moreover, having established these concepts, it is then possible in Chapter 14 to consider string data more closely, thereby revealing the underlying representation that was concealed by the abstract definition. Chapter 16 integrates the fundamental data structures with the construction of a data-driven teaching machine, which is the most sophisticated example of programming presented in the text.

Chapter 17 includes three important topics that often appear in the first programming course: recursion, abstract data types, and analysis of algorithms. At Stanford, which is on the quarter system, we teach all these topics in the second course. If you decide to teach recursion in the first course, I strongly recommend that you do so early enough to allow students time to assimilate the material. One possibility is to discuss recursive functions immediately after Chapter 5 and recursive algorithms after Chapter 6. Another approach is to cover recursion and analysis of algorithms together at the end of Chapter 12.

INSTRUCTOR'S MANUAL

The Instructor's Manual contains supplemental materials including a course syllabus, suggestions for lecture design, sample assignments and examinations, and solutions to all programming exercises. In addition to the printed manual, instructor's who adopt this text can retrieve electronic copies of solution sets and related materials. For details on obtaining solutions, please contact your local Addison-Wesley

representative. All other supplemental material is available on-line.
For explicit instructions see Appendix B.