

# **A Balanced Introduction to Computer Science**

**David Reed                      ©Prentice Hall, 2004**

## **Chapter 0. Introduction and Overview**

Welcome to *A Balanced Introduction to Computer Science*. There are a number of reasons why you might be reading this text. Perhaps you have had limited experience with computers and would like to know more about how they work and how to control them. Or perhaps you recognize the marketability of programming and computer literacy, and would like to expand your skills for the future job market. Or perhaps you are just curious about the World Wide Web and want to know what all the hype is really about. In any case, you are embarking on what I hope will be an exciting, challenging, and rewarding experience.

### **Balancing Breadth and Depth**

This text is different from most introductory computing texts in that it attempts to maintain a balance between computing breadth and programming depth. Traditionally, introductory texts have focused almost exclusively on one approach or the other. Breadth-based texts have emphasized a broad understanding of computers and computer science. By surveying a wide range of topics such as computer organization, graphics, networking, and technology in society, the intent is for students to experience the breadth of the field and develop a perspective to later understand and appreciate the role of technology in their lives. Alternatively, depth-based texts have focused more deeply on the role of programming in computing. The discipline of programming not only develops problem-solving skills, but also is central to many areas of computer science and thus important to appreciating their significance.

While each of these approaches has merit, there are potential weaknesses to either extreme. A breadth-based survey of computing can be too superficial, presenting a broad perspective to students who lack the context or experience to fully comprehend it. And while programming depth can provide experience with many computing concepts, developing proficiency as a programmer and problem-solver requires extensive hands-on experience (especially when learning a complex language such as C++ or Java), and may not be directly relevant to all students.

The approach taken by this text is to balance breadth and depth. Chapters are included on concepts and issues in computing that are most relevant to the beginning student,

including computer terminology, the Internet and World Wide Web, the history of computing, the organization and manufacture of computer technology, and technology's impact on society. Mixed among these breadth topics are chapters that introduce fundamental programming concepts and skills in a hands-on, tutorial format. Using the programming language JavaScript, students will develop skills in designing and implementing interactive Web pages. JavaScript's simplicity, natural interfaces, and seamless integration with the Web make it possible for novices to develop interesting and engaging programs quickly. In addition, JavaScript is always available (for free!) for anyone with a Web browser, making it easy to apply programming skills learned from this text to everyday problems.

In striking a balance between breadth and depth, the intent of this text is not to be a complete and exhaustive survey of computing or a reference on JavaScript. Breadth chapters focus on key ideas and concepts that are relevant to beginning students as they attempt to understand computing technology and the field of computer science. Likewise, programming chapters focus on JavaScript features that demonstrate fundamental programming concepts while also allowing for interesting and engaging applications. Links to other sources are provided for the interested reader, including supplemental material and exercises at the end of each programming chapter. This provides a broad perspective on computing as well as enough problem solving and programming depth to appreciate the significance of computer science.

## **Text Goals**

There are three main goals to this text and its accompanying resources. First, it serves to expose the student to the breadth that is the field of computer science. Computer science is more than just the study of computers – it focuses on all facets of computation, from the design and analysis of algorithms (step-by-step sequences of instructions for carrying out tasks), to the engineering and manufacture of computer components, to the development of software systems. Through readings and the use of online resources, the student will study topics such as the history of computer technology, the underlying architecture of modern computers, the translation and execution sequence of programs, and the capabilities and limitations of computation. Using software simulators, the student will build virtual components of a computer and watch the flow of information as a program is translated and executed on the low-level machinery. Through this combination of reading and experimentation, hopefully these concepts will come alive for the student and provide a sense of what computer science is all about.

The second main goal of this text is to teach the student the fundamentals of programming. *Programming is the process of solving problems on the computer*, that is, devising solutions to specific tasks and formalizing those solutions in a language the computer can understand and execute. Programming is the central activity in computer science, providing an inroad to many of the interesting facets and challenges of the field. In learning to program, the student will be learning to analyze problems, think logically,

formalize his or her thoughts, and solve problems. It is a discipline, since a systematic approach must be learned, but it is also a creative process, since novel approaches must be found to attack new problems. And since many of the skills developed in programming apply to problem solving in general, experience gained through this text should carry over to other disciplines as well.

The third main goal of this text is to demonstrate the scientific and interdisciplinary nature of computing. Research in various fields of study, most notably the mathematical and natural sciences, is becoming increasingly dependent on computers and programming. By studying and investigating applications in fields such as biology, physics, psychology, and even economics, the student will learn to apply his or her programming skills to a wide range of problems. In addition, the student will develop empirical skills that are common to all scientific endeavors.

## **Text Features**

The balanced approach to computer science and programming taken by this text is evident in the layout of its chapters. There are two types of chapters in the text, those that use narrative to introduce key concepts of computing (i.e., the computer science breadth chapters: 1, 3, 6, 8, 10, 12, 14, 16, and 18) and those that use a tutorial style to developing problem-solving and programming skills (i.e., the programming depth chapters: 2, 4, 5, 7, 9, 11, 13, 15, and 17). The interleaving of these chapters is both intentional and important. It provides variety in the types of activities students will undertake, and thus may be more accommodating to students with different learning styles. Readings and classroom discussions serve as buffers between programming tutorials, allowing the student more time to assimilate programming concepts and skills before beginning the next tutorial. Finally, and perhaps most importantly, the interleaving of chapters supports the student's understanding and appreciation of the content. For example, after developing their own home pages in Chapter 2, students are better prepared to understand what the Web is and how it works in Chapter 3.

Features of the computer science breadth chapters:

- They focus on topics that are most relevant to a beginning student. The goal is not to inundate the student with details, but instead to emphasize the central ideas of that topic.
- Illustrations are used whenever possible to illuminate key points.
- Web-based visualization tools (accessible at the book's Web page: <http://www.prenhall.com/reed>) are provided to complement many of the chapters and support active learning. For example, chapter 14 integrates a suite of simulators that allows the student to explore the internal workings of computers.
- Each chapter ends with Review Questions that encourage reflection and the integration of content from that chapter.

Features of the programming depth chapters:

- They are presented in a tutorial style, recognizing that the only way to learn programming (and, more generally, problem solving) is to actually do it.
- Exercises follow an incremental approach, allowing students to master programming concepts by first studying existing programs (which are accessible at the book's Web page: <http://www.prenhall.com/reed>) and then making modifications using new tools and constructs. Eventually, students create new programs for solving interesting and hopefully engaging problems.
- Common errors and points of confusion are identified and discussed in special sections called "Common errors to avoid..."
- Problem-solving and program design advice are provided in special sections called "Designer secrets..."
- Each chapter includes a Chapter Summary that presents the key concepts and programming tools in a concise bullet list.
- Supplemental material and exercises are provided at the end of each chapter for further study.

Appendices are provided at the end of this text as references. Appendices A and B provide tutorials on Web browsers and common text editors, which may be useful for students who are not already familiar with computers and the Web. Appendix C is an HTML reference, collecting all of the HTML elements used throughout the text in a table. Appendix D is a JavaScript reference, which similarly collects all of the JavaScript programming constructs. Appendix E provides a full listing of the `random.js` library, which is introduced in Chapter 5 and used in subsequent chapters.

Finally, a collection of nine laboratory assignments, each corresponding to a programming depth chapter, is available to supplement this text. These lab assignments emphasize experimentation, analysis, and the use of programs for solving interdisciplinary problems.

## **Advice for the Student**

This text does not assume any previous experience with computer applications or programming. Certainly, familiarity with computers is a plus (e.g., word processing or email), but is not necessary. Basic computer terminology will be covered in the text as needed, and appendices are provided to assist the novice with simple computer skills, such as using a text editor, saving files to a disk, and browsing the Web. The goal is not to teach you everything you could ever want to know about computers and programming, but instead to provide you with a working set of skills and knowledge. Whenever possible, links to further readings will be provided in case you are interested in a topic and would like to learn more.

Through the use of readings, exercises, and experiments, this text aspires to provide you with a broad sense of what computer science is all about, while simultaneously developing depth as a problem-solver and programmer. The choice of JavaScript as the

medium for developing programming skills was specifically made to make this task simpler and also more relevant to students. JavaScript was designed to be a simple scripting language for controlling pages on the World Wide Web. Using JavaScript, you can control actions with the click of the mouse or generate dynamic images on a page. As such, learning JavaScript opens the door for many exciting applications on the Web. Similarities between JavaScript and the programming languages Java and C++ also mean that experience with JavaScript programming can be a stepping-stone to larger-scale programming in these industry-strength languages.

Whether you choose to continue studies in computer science, or merely wish to apply computing skills to your everyday life, the balanced coverage of computing topics as found in this text should prove valuable to you. As always, enjoy and learn.

### **Advice for the Instructor**

The layout of the chapters in this text is designed to provide maximum flexibility for the instructor. Depending on the preferences of the instructor and the goals of the particular course, the right balance and order of the material can be determined for your needs. If you are teaching a traditional non-majors course, then a roughly even balance between breadth and depth probably makes sense. If the students are computer knowledgeable and taking this course as part of a computer science sequence, then some breadth topics may be skipped or shortened to allow for greater programming depth.

For example, the materials in this text have been used in a non-majors course at Creighton University. Since the majority of the students in this course are not planning to become computer science majors, the emphasis in the course is in providing a balance between essential topics and fundamental programming concepts. As such, there is a roughly 50/50 division between breadth and depth. Only the first six programming chapters (Chapters 2, 4, 5, 7, 9, and 11) are covered, each taking up one week of class time. While this may seem limiting (no loops?!), the material in these chapters is sufficient to impart the flavor of programming and also to allow the students to do interesting and engaging things. Mixed between these programming chapters are most of the breadth chapters, each taking up one or two 50-minute periods for lecture and/or discussion. Not all of the breadth chapters from the text are covered each semester, and the order may vary as well. For example, in a recent semester I skipped Chapter 8 and moved the content of Chapter 12 later in the course. In addition, I integrated four laboratories into the class periods, which served to apply the students' programming and problem-solving skills to interdisciplinary problems.

By contrast, these materials have been used in a very different course at Dickinson College. This particular course is the first in a two-course sequence that fulfills a laboratory science requirement for the college, and is also considered the first course in the computer science major. As such, greater expectations in terms of time and effort are expected of the students. Weekly two-hour lab periods accompany the class, in which

students complete lab assignments. Programming chapters are covered in two to three 50-minute periods each, and select breadth chapters are covered in one to two 50-minute periods.

One of the strengths of this book is the flexibility that it provides the instructor. An instructor might choose one of these extremes, or select a different balance of breadth and depth to best suit his or her students. For example, I envision that many instructors might choose to cover Chapter 13 (loops), perhaps substituting for in-class laboratories or certain breadth topics. The interleaving of breadth and depth chapters throughout the text is specifically designed to support the content and also vary the types of learning activities students engage in. While the relative order in which the programming depth chapters are covered is constrained by their content, it is certainly possible to omit or move some of the computer science breadth chapters. For example, some instructors might prefer covering the history of computers (Chapter 6) earlier in the course, or perhaps covering the chapters on how computers work (Chapter 14) and are built (Chapter 16) together.

As always, enjoy, teach, and learn.

## **Acknowledgements**