Programming by Design

A First Course in Structured Programming

PHILIP L. MILLER and LEE W. MILLER

with Purvis M. Jackson

SPECIAL EDITION

Programming by Design

A First Course in Structured Programming

Special Edition

Philip L. Miller, PhD
Computer Science Department
Carnegie Mellon University

Lee W. Miller
Computer Science Department
Carnegie Mellon University

with Purvis M. Jackson
Software Engineering Institute
Carnegie Mellon University

Wadsworth Publishing Company
Belmont, California

Carnegie Publishing, Inc.
Pittsburgh, Pennsylvania

Copyright © 1987, 1986 by Carnegie Publishing, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the copyright holder.

Portions of Chapter 2 of this book are from *Karel The Robot* by Richard E. Pattis, copyright (c) 1981, John Wiley & Sons, Inc. Reprinted with permission.

ISBN 0-534-08244-0

Printed in the United States of America
2 3 4 5 6 7 8 9 — 90 89 88 87

Foreword

The driving force behind the creation, introduction, and refinement of the College Board's Advanced Placement (AP) Computer Science course has been Phil Miller. Far more than any other one person, Phil has shaped both the spirit and content of that course. He served first as a member of a College Board Task Force considering College Board offerings in computer science, then as a member of the initial AP Development Committee for the AP Computer Science examination, and finally as Chairman of that committee for the last 4 years. Now Phil, together with Lee Miller, has authored a textbook that displays the same principles of programming methodology that he has consistently advocated for the AP Computer Science course since its inception.

As an observer of the work of the AP Computer Science Development Committee and as a reviewer of *Programming by Design*, I see the same themes dominating the textbook that I have seen driving the AP course. The use of Karel the Robot in the textbook as a precursor to Pascal mirrors the influence of Phil on the AP course, where the earliest possible introduction of procedural abstraction as a dominant characteristic is one of the principles on which Phil's views prevailed within the AP Development Commmittee even before he became its chairman.

The exact course material in a first course varies from university to university. It is my observation through work with the AP Computer Science Committee and the GRE Computer Science Committee that this text is in line with what is being taught at leading computer science departments around the country, though I have made no systematic investigation of curricula. This text solidly covers the aspects of programming methods of the year-long AP Computer Science course, though teachers will want to supplement *Programming by Design* with a good algorithms/data-structures text for a year-long AP course.

The effervescence of Phil Miller's personal style so evident in earlier versions of this text has been toned down in this published edition. I personally preferred the "rough-cut," intimate approach, but appreciate the need for a more civilized touch. Reviewing this text is much like watching Phil in a three piece suit delivering a polished lecture on the AP program after having watched him make the same points in committee using much more colorful language while he was dressed in blue jeans. Nevertheless, his commitment to the teaching of programming methods by means of real communication with the student still comes through. *Programming by Design* is not a dry presentation of programming methodology nor is it a passive instrument in the teacher's hand. It is rich in pedagogical material and tries to teach by engaging the student, and I believe it succeeds.

J. R. Jefferson Wadkins, Senior Examiner
Test Specialist for the AP CS Examination
Educational Testing Service

Preface

When we set out to write this book, we had one simple goal in mind—to develop a text that would present the materials appropriate for a first course in computer science. We were motivated to take on the project by the fact that—even though there were numerous texts on the market—none of the available texts effectively supported the aims of the introductory courses offered at Carnegie Mellon University. No matter which of the available texts we tried, we found ourselves faced with two primary problems: (1) topics we hold to be important were not treated, and (2) far too much of our time in lectures had to be devoted to explaining to students the material they had read from the textbook. In attempts to overcome those two problems, we developed a significant stack of lecture notes and extended examples to explain further the concepts we found to be lacking or inadequately explained in each of the texts we had attempted to use. Increasingly, we found ourselves relying more on our notes and less on available texts. Eventually, we arrived at a point where we began distributing our materials to the students, which enabled us to follow the course we felt to be appropriate. Moreover, it enabled us to benefit from the feedback supplied by the students and other instructors—feedback that told us that although we had made significant progress toward solving our first problem, the second problem was still evident.

The feedback we received made poignant the need for a *thorough* text, one that would go beyond the ritual of *discussing* a topic to the unorthodox practice of *explaining* the topic. Toward that end, we have attempted to develop a book that incorporates the *teaching* we had been forced to add to texts we previously used. This meant adding hundreds of illustrations and hundreds more programming examples. Our experience with previous versions of this book suggests that we have gotten a great deal of the teacher into the text. This book is intended now for use at either the college level or in high schools that offer advanced courses. No background in computing or advanced mathematics is assumed. The only prerequisite is literacy in the English language.

To Teachers

Although we intend this book to be used to teach programming methodology, we realize that programming skills must be learned by writing programs in a particular language. Toward that end, our presentation includes the study of two programming languages, Karel and Pascal, both of which are used as vehicles for developing general programming and problem-solving skills. Pascal is given the more thorough treatment of the two.

There are numerous reasons for selecting Pascal as a teaching language. Most important is the prevalence of computer systems supporting Pascal, the pertinence of the high-level, block-structured features of the language, our own success with it at Carnegie Mellon University, and the strong endorsement it has received from the College Board's committee that designed the Advanced Placement Computer Science (APCS) course.

Having found that the initial segment of a programming course is crucial to students' perception of the subject, we have selected the language Karel as a precursor to Pascal. We have found that Karel enables them to grasp somewhat easily the concepts of structured programming, which we later look at in more detail when discussing Pascal. We use Karel as an overview of the subject; we use Pascal to provide the necessary detail and reinforcement. With Karel, the novice can plunge into programming and problem solving with a minimum of overhead. Further, Karel provides an interesting problem domain, within which students can learn to write increasingly complex, well-structured programs.

Karel is a robot simulator language, developed by Richard E. Pattis, that allows students to see their programs execute in the two-dimensional world of Karel the Robot. At Carnegie Mellon, we teach

此为试读,需要完整PDF请访问: www.ertongbook.com

Karel with the aid of the simulator software; however, others have reported success in using Karel without the simulator, i.e., by having students develop their programming solutions on paper. In our own experience, we have found that the time spent with Karel pays for itself many times over. It provides a very accurate overview of structured programming methodology. More importantly, it makes subsequent study of Pascal much easier for the student. Beyond this, thanks to the intuitive nature of Karel, it is superb in overcoming the "fear of computing" syndrome common to many students.

To Students

A common, and understandable, question many students ask is, "Why should I learn about computers and programming?" There are, of course, a number of ways that question might be answered. In general, however, there are four reasons why we think you should learn about computing.

Computers have become very prevalent in today's society. The computer has already changed, or is in the process of influencing, many aspects of our lives, ranging from the scientific exploration of space to the cash registers at the local supermarket. Every time we pick up the telephone or watch the evening news, we witness applications of computing. With computers so prevalent, it is important for you to understand the principles of computing and how they affect your life. Usually, people feel less annoyed and less threatened by things they understand. Thus, the first reason for understanding computing is to better understand the world around us.

Computing is thought of by many people to be the province of the scientifically inclined, the folks who love and live by numbers. Today, however, computers are no longer relegated to the laboratories and offices of engineering and science departments. In fact, they are used throughout the arts, business, and the humanities. In each of these areas, computing is allowing new approaches to long-standing problems. Thus, a second reason for learning about computing is to share in the intellectual stimulation it can foster—in any discipline.

Estimates by experts show that there were 100,000 available positions for software professionals in 1980 that simply could not be filled because the demand for software far exceeds our ability to produce software professionals. The shortage is expected to reach one million by 1990, if the current trend continues. Unlike people educated or trained in other professions, software professionals are virtually guaranteed well-paying job opportunities. Thus, the third reason for learning about computing is to gain the knowledge and experience that can lead to a lucrative career within the computing industry.

The old adage that states "Time is money" can be altered slightly to "Time is life" to more accurately reflect the importance of time. How we spend our time is how we spend our lives. All too often we literally waste time by doing the same things over and over that we could do more effectively in other ways. The average person spends a significant amount of time on relatively mundane tasks—such as record keeping and filing expenses—that can be handled more quickly and more accurately by computer. Thus, the fourth—and perhaps most important—reason for learning about computing is to gain more control over how we spend our time.

The material presented in this book will provide you with the principles and concepts necessary for you to create a whole range of programs capable of solving a number of important problems. But beyond that, you can apply many of the principles to problems other than those involving programming. In short, the concepts in this text provide methods that you may use to think about any complex problem or situation. Properly used, they will serve you well. Work the exercises and solve the problems at the end of each section to make sure you understand the principles.

Some of the sections and exercises are marked with asterisks (*) to indicate that you may wish to skip them on first reading. We suggest you skip all sections marked with three asterisks (***) on first reading. Some readers may wish to skip these items altogether.

To All

This book comprises 18 chapters, 8 appendices, and 4 indices. It contains nearly 500 illustrations and over 3,000 index entries. We utilized the computer to prepare every facet of this book. All page layout was done using the computer; all of the hundreds of illustrations were prepared by computer and merged electronically using custom-made software; and the thousands of index entries were processed completely automatically. We can not imagine preparing a document such as this without a computer.

Whatever success this book, or subsequent editions of it, may have will be due in large part to the good ideas and sound advice offered by our colleagues at Carnegie Mellon University and elsewhere in the Computer Science community. Were it not for their suggestions and continued encouragement, this book would not have reached its current state. For their support throughout this project, we wish to thank all of the people who read and commented on drafts of chapters, who suggested exercises or problems, and who gave to us the inspiration to continue. We especially thank the lecturers at Carnegie Mellon: Terry Gill, Nahid Capell, Dennis Goldenson, Jim Roberts, Jacobo Carrasquel, and particularly Mark Stehlik, whose genuine concern for students has found its way onto a great many pages of this book. We also thank Rob Chandhok, Harry Holland, Wanda Keppler, Becky Alden, Michelle Lurye, Eric Goodman, Amy McMurtry, and Nick Spies. We recognize the students who suffered with us through earlier drafts while we developed and refined this work. We would like to especially thank Bob Spies, who patiently provided many hours of technical and other assistance at a time when he was quite busy with many other projects.

Although the first draft of this book was written by a two-person author team, the final form was the result of three people. The extensive contributions of Purvis Jackson can be seen on nearly every page of this book. To recognize his contributions, we include his name in the appropriate place, on the front cover.

P.L.M.

L.W.M.

Pittsburgh, Pennsylvania
1986

Contents

Chapter 5: Details of Input, Output, and Variables 139

Chapter 6: Conditional Execution 175

此为试读，需要完整PDF请访问：www.ertongbook.com

Chapter 9: Parameters 273

Chapter 10: Functions 301

Chapter 11: Recursive Subprograms 317

UNIT IV: User-Defined Types 335

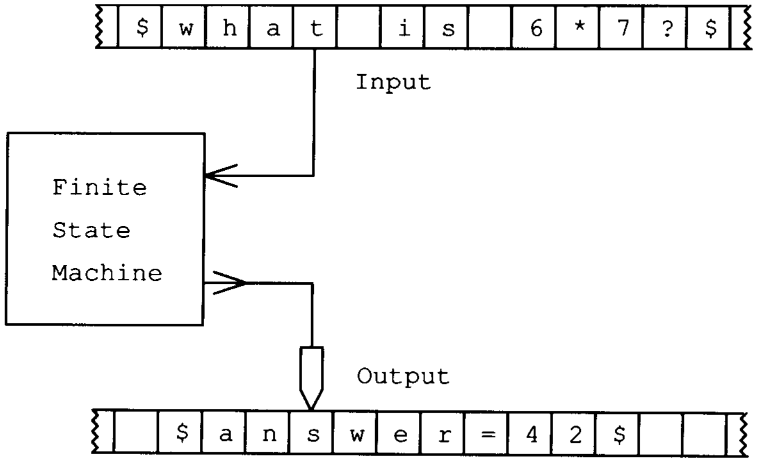
Chapter 12: Simple Types 337

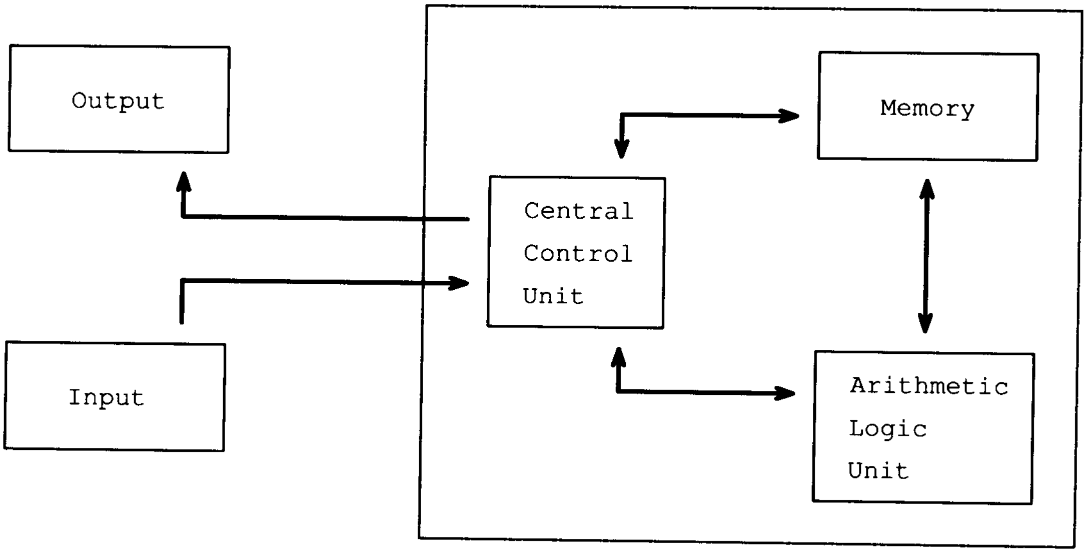
Chapter 13: Structured External Types 357

UNIT I: Foundations of Computing

Chapter 1: An Historical Perspective

Chapter 2: Overview of Programming





此为试读，需要完整PDF请访问：www.ertongbook.com