

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 Committee 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

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

UNIT I: Foundations of Computing 1

Chapter 1: An Historical Perspective 3

1.1. Automatic Computing	3
Charles Babbage	4
Early American Computers	5
1.2. Theoretical Underpinnings	6
David Hilbert	7
Kurt Gödel	7
Alan Turing	7
Alonzo Church	8
1.3. Modern Computing	8
The Von Neumann Machine	8
The Modern Computer	10
Programming Languages	10
The Personal Computer	14
1.4. Areas of Computer Science	16
Computing Systems	16
Programming Systems	17
Artificial Intelligence	18
Theory	18
1.5. Summary	19

Chapter 2: Overview of Programming 21

2.1. Programming	21
2.2. Programming Languages	21
2.3. The Robot World	22
Karel's World	22
Karel's Capabilities	22
Tasks and Situations	24
2.4. Primitive Instructions and Simple Programs	25
Changing Position	25
Handling Beepers	27
Finishing a Task	27
A Complete Program	27
Error Shutoffs	30
Programming Errors	31
Problem Set	32
2.5. Extending Karel's Vocabulary	36
Creating a More Natural Programming Language	36
A Mechanism that Defines New Instructions	36
Block Structuring	37
The Meaning and Correctness of New Instructions	38
Defining New Instructions in a Program	39
Boxing: How Karel Understands a Program	41
An Ungrammatical Program	42
Programming by Stepwise Refinement	43
Writing Understandable Programs	47
Problem Set	48
2.6. Conditionally Executing Instructions	51
The IF/THEN Instruction	51
The Conditions Karel Can Test	51
Simple Examples of the IF/THEN Instruction	52
The IF/THEN/ELSE Instruction	56

Nested IF Instructions	58
Transformations for Simplifying IF Instructions	60
The Dangling ELSE	63
Problem Set	64
2.7. Instructions That Repeat	66
The ITERATE Instruction	67
The WHILE Instruction	68
Repeating Instructions and Block Structure	72
IF Instructions in WHILE Loops	74
A Large Program Written by Stepwise Refinement	76
Problem Set	85

UNIT II: Elements of Pascal 91

Chapter 3: Programming in Pascal 93

3.1. Comparing Pascal with Karel	93
3.2. Generating Output	95
3.3. Defining New Statements	96
Questions	98
Exercises	98
Problems	98
3.4. Naming Constants	99
3.5. Writing Expressions	101
3.6. Storing Expressions	102
Questions	104
Exercises	104
Problems	105
3.7. Commenting Programs	105
3.8. Formatting Output	105
3.9. Labeling Output	106
3.10. Solving Bigger Programming Problems	106
3.11. Reading Input	108
3.12. Prompting for Input	108
3.13. Naming All Constants	109
3.14. Picking Good Names	110
Questions	110
Exercises	111
Problems	111
3.15. Putting It All Together	111
3.16. Chapter Summary	113
3.17. Summary of Style	115
3.18. Summary of Terms	115
3.19. Chapter Problems	115

Chapter 4: Background Tools 117

4.1. A Tool for Describing Pascal and Other Languages	117
Questions	119
Exercises	119
4.2. Operators and Expressions	120
Questions	122
4.3. Pascal's Four Primitive Types	122
INTEGER	122
REAL	125
CHAR	127
BOOLEAN	129
4.4. Some Finer Points of Operators, Types, and Evaluation	131
Binary and Unary Operators	131

Precedence Rules	132
Type Coercion	135
Type Cardinality	135
4.5. Chapter Summary	136
4.6. Summary of Terms	136
4.7. Chapter Questions	137

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

5.1. The Details of Output	139
The Output Display	139
The Literal String and the Single Quote	140
Writing Output	140
Output Format Specification	141
Writing Multiple Expressions	143
When Expressions Are Actually Written	144
Questions	144
Exercises	145
Problems	145
5.2. Variables and Assignment	146
The Variable	146
The Assignment Statement	153
Questions	157
Exercises	157
Problems	158
5.3. The Details of Input	159
READ and READLN	159
Reading Multiple Values	164
When Values Are Actually Read	167
Common Problems with Reading Input	167
Questions	168
Exercises	168
Problems	170
5.4. Chapter Summary	170
5.5. Summary of Style	172
5.6. Summary of Terms	172
5.7. Chapter Problems	173

Chapter 6: Conditional Execution 175

6.1. IF-THEN Statement	175
Formal Definition	176
Compound Statements	176
Pascal's Conditions	177
Questions	178
Exercises	179
Problems	179
6.2. IF-THEN-ELSE Statement	180
Formal Definition	181
Dangling ELSE	181
Questions	182
Exercises	182
Problems	183
6.3. CASE Statement	183
Formal Definition	188
Questions	189
Exercises	189
Problems	189
6.4. A Debugging Aid	190
6.5. Simplifying the Condition	191
Commutative Law	191
Associative Law	192

Distributive Laws	192
DeMorgan's Laws	193
Other Laws	193
Questions	194
Exercises	195
6.6. Chapter Summary	195
6.7. Summary of Style	197
6.8. Summary of Terms	197
6.9. Chapter Problems	198
 Chapter 7: Repetition	 199
7.1. The WHILE Statement	200
Formal Definition	206
Infinite and Nonexecuting Loops	206
Questions	206
Exercises	207
Problems	207
7.2. The FOR Statement	207
Formal Definition	210
Nested Loops	211
Questions	219
Exercises	219
Problems	220
7.3. The REPEAT Statement	221
Loop Equivalence	222
Formal Definition	222
Questions	222
Exercises	223
Problems	223
7.4. Loop Construction and the Loop Invariant	224
Method Not Magic	224
A Method of Determining Which Loop Construct to Use	224
A Method of Selecting the Loop Condition	225
A Method of Completing the Loop Body	226
Adding Requirements Outside the Loop	226
A Method of Verifying Loop Termination	227
A Method of Verifying That a Loop Works	227
A Method of Verifying That a Loop Always Works	228
Method for Building Loops	228
Applying the Systematic Approach to a Familiar Problem	229
Systematic Approach to Another Problem: The Marble Bag	231
Questions	232
Exercises	233
Problems	233
7.5. Chapter Summary	233
7.6. Summary of Style	235
7.7. Summary of Terms	236
7.8. Chapter Problems	236

UNIT III: Modular Programming 239

Chapter 8: Procedures 241

8.1. Modular Programming	241
Problem Solving	241
Modular Problem Solving	241
Top-Down Design and Solution Trees	242
A Programming Methodology	244

Questions	245
Exercises	245
8.2. Subprogram Concepts	245
A Model of Subprogram Execution	246
Subprogram Declarations and Calls	248
Procedures and Functions	248
Questions	248
8.3. Simple Procedures	249
Subprogram Comments and Declarations	251
8.4. Scope of Identifiers	251
Local and Global Declarations	252
Boxing Programs	252
Simulating Programs	254
Up-Level Addressing and Side Effects	263
Nested Declarations	264
A Style of Declaration	267
Questions	268
Exercises	269
8.5. Chapter Summary	269
8.6. Summary of Style	270
8.7. Summary of Terms	271
8.8. Chapter Problems	271

Chapter 9: Parameters 273

9.1. Value Parameters	273
Declaring Procedures with Value Parameters	275
Formal Parameters, Actual Parameters, and Parameter Binding	277
Expressions as Actual Value Parameters	279
Procedures with More Than One Value Parameter	282
Up-Level Addressing of Variables: A Practice To Be Avoided	284
Questions	285
Exercises	286
Problems	287
9.2. Variable Parameters	287
Declaring Procedures with VAR Parameters	289
A Technique of Ensuring Parameter Passing	292
Subprogram Comments: Another Tool in the Toolbox	295
A Style of Declaration	295
Questions	296
Exercises	297
9.3. Chapter Summary	298
9.4. Summary of Style	299
9.5. Summary of Terms	300
9.6. Chapter Problems	300

Chapter 10: Functions 301

10.1. What Functions Are	301
10.2. Predefined Functions	303
10.3. Calling Functions	303
10.4. Defining New Functions	304
10.5. When To Use Functions	308
10.6. When Not to Use Functions	309
10.7. Structured Comments for Functions	310
10.8. Programming with Stubs	310
Questions	312
Exercises	313
Problems	314
10.9. Chapter Summary	315
10.10. Summary of Style	316
10.11. Summary of Terms	316

10.12. Chapter Problems	316
Chapter 11: Recursive Subprograms	317
11.1. Recursive Functions	318
11.2. Recursive Procedures	327
Questions	333
Exercises	333
Problems	333
11.3. Chapter Summary	334
11.4. Summary of Terms	334
 UNIT IV: User-Defined Types	 335
Chapter 12: Simple Types	337
12.1. The Need for Other Types	337
12.2. Defining a New Type	340
12.3. A Taxonomy of Pascal Types	340
Questions	343
12.4. Enumerated Types	343
One More Example	345
Order	345
Ordinal Types	347
Input and Output of Enumerated Types	347
Some Fine Points	348
Questions	350
Exercises	350
12.5. Subrange Types	351
Questions	353
Exercises	353
12.6. Chapter Summary	354
12.7. Summary of Terms	355
 Chapter 13: Structured External Types	 357
13.1. An Overview of Files	357
13.2. Pascal Files	358
Declaring Files	360
The Most Common Files: Text Files	360
Peculiarities of Pascal File Variables	361
Questions	362
Exercises	362
13.3. Reading Files	363
Checking for the End-Of-File Marker	364
Checking for End-Of-Line	365
Looking Ahead: The File Buffer Variable	368
Counting Lines: Another Application	369
Common Errors	369
Questions	370
Problems	370
13.4. Writing Files	371
13.5. Internal Files, External Files, and the Program Heading	372
Questions	373
Problems	373
13.6. Selected Uses of Files	374
Displaying a Text File	374
Merging Two Files	375
13.7. Chapter Summary	376

13.8. Summary of Terms	378
13.9. Chapter Problems	378

Chapter 14: Arrays 381

14.1. The Need for Arrays	381
14.2. Array Terminology and Declaring Arrays	382
14.3. Manipulating Arrays and Their Components	383
14.4. Some Important Points About Arrays	385
Random Access	385
Writing an Entire Array	385
Copying an Entire Array	385
Finding the Maximum Element	386
14.5. Other Kinds of Array Indexing	387
14.6. Arrays of Arrays and Multiple Indexing Sets	389
14.7. String Variables	392
Initializing String Variables	393
Reading String Variables	394
Writing String Variables	394
Comparing String Variables	394
14.8. Questions	395
14.9. Problems	396
14.10. Chapter Summary	396
14.11. Summary of Terms	397

Chapter 15: Records and Sets 399

15.1. Records	399
Declaring Records	401
Accessing Record Components	402
Compact Record Definitions	402
Collections of Records	403
The WITH Statement	404
A Different View of Records	405
Questions	406
Exercises	407
15.2. Sets	407
Set Definition	407
Set Operations	407
Set Relations	408
The Empty Set and the Universal Set	409
Pascal Sets	410
Questions	413
Problems	414
15.3. Solving a Larger Problem: Managing a Baseball Team	415
Problem Description	415
Program Design and Representation	415
15.4. *** Variant Records	420
15.5. Chapter Summary	423
15.6. Summary of Terms	424

UNIT V: Modern Programming 425

Chapter 16: Style: Writing Better Programs 427

16.1. A Style of Program Specification	427
An Example Functional Specification	428
16.2. A Style of Program Design	428
Top-Down Design	429

Example of Program Design	429
16.3. A Style of Program Implementation	430
Declarations	431
Inter-Module Communication	432
Comments and Documentation	433
Efficiency	436
Readability	437
Understandability	440
An Example of Program Implementation	441
16.4. A Style of Program Verification	444
16.5. Notes on Debugging	446
 Chapter 17: Dynamic Memory Allocation	 449
17.1. Allocation of Computer Memory	449
Questions	451
17.2. Manipulating Dynamic Variables	451
Questions	456
Exercises	456
17.3. Dynamically Allocating Record Variables	457
Questions	458
17.4. Linking Dynamic Variables	459
An Analogy to the Linked List	459
A Linked List in Pascal	463
An Implementation of a Linked List in Pascal	467
Replacing an Array of Pointers with a Linked List	468
Questions	470
17.5. Chapter Summary	470
17.6. Summary of Terms	471
17.7. Chapter Problems	471
 Chapter 18: Searching and Sorting	 473
18.1. Searching	473
Sequential or Linear Search	474
The Ordered Search	476
Binary Search	477
Comparing Algorithms	478
Families of Functions	479
Order Arithmetic and Order Comparisons	481
Comparing Searching Algorithms	482
Summary of Searching Algorithms	482
Questions	483
Exercises	483
18.2. Sorting	484
Order, Order!	484
Internal and External Sorting	485
Insertion Sort	485
Bubble Sort	487
Selection Sort	488
Merge Sort	490
A Lower Bound on Comparison-Based Sorting	493
Selecting the Best Sorting Algorithm	494
Questions	495
Problems	495
18.3. Chapter Summary	496
18.4. Summary of Terms	497

Appendix I. Karel Reserved Words and Special Symbols	499
Appendix II. Karel Syntax—BNF	501
Appendix III. Karel Syntax—Bubble Diagrams	503
Appendix IV. Pascal Reserved Words and Special Symbols	507
Appendix V. Pascal Syntax—BNF	509
Appendix VI. Pascal Syntax—Bubble Diagrams	517
Appendix VII. Pascal Library Routines	543
Appendix VIII. Character Sets	545
References	547
Index of Figures	549
Index of Tables	551
Index of Programming Examples	553
Index of Terms	559

UNIT I: Foundations of Computing

Chapter 1: An Historical Perspective

Chapter 2: Overview of Programming

