

Texts in Computer Science

Series Editors

David Gries, Department of Computer Science, Cornell University, Ithaca, NY, USA

Orit Hazzan , Faculty of Education in Technology and Science, Technion—Israel Institute of Technology, Haifa, Israel

Titles in this series now included in the Thomson Reuters Book Citation Index!

'Texts in Computer Science' (TCS) delivers high-quality instructional content for undergraduates and graduates in all areas of computing and information science, with a strong emphasis on core foundational and theoretical material but inclusive of some prominent applications-related content. TCS books should be reasonably self-contained and aim to provide students with modern and clear accounts of topics ranging across the computing curriculum. As a result, the books are ideal for semester courses or for individual self-study in cases where people need to expand their knowledge. All texts are authored by established experts in their fields, reviewed internally and by the series editors, and provide numerous examples, problems, and other pedagogical tools; many contain fully worked solutions.

The TCS series is comprised of high-quality, self-contained books that have broad and comprehensive coverage and are generally in hardback format and sometimes contain color. For undergraduate textbooks that are likely to be more brief and modular in their approach, require only black and white, and are under 275 pages, Springer offers the flexibly designed Undergraduate Topics in Computer Science series, to which we refer potential authors.

More information about this series at <https://link.springer.com/bookseries/3191>

Marco T. Morazán

Animated Problem Solving

An Introduction to Program Design
Using Video Game Development

 Springer

Marco T. Morazán
Department of Computer Science
Seton Hall University
South Orange, NJ, USA

ISSN 1868-0941 ISSN 1868-095X (electronic)
Texts in Computer Science
ISBN 978-3-030-85090-6 ISBN 978-3-030-85091-3 (eBook)
<https://doi.org/10.1007/978-3-030-85091-3>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To my parents, Doris and Marco, who taught me to love teaching and to realize that having an education is not a privilege but a responsibility.

Preface

Everybody engages in problem solving. It is a natural and inevitable part of life. Historically, the link between problem solving and programming has been less emphasized. When you write an essay, you are programming—at different levels many times. You make sure ideas flow and arguments make use of the data you are analyzing. You write several drafts of the essay. Each draft represents a refinement. Every paragraph has a point and you avoid repeating yourself. All this is part of programming, including computer programming. Programmers, people who solve problems using a computer, go through the exact same steps to write a program. The same steps are taken by a psychologist analyzing a patient and by a chemist experimenting in a laboratory. Even a painter engages in programming. No? Does a painter not want to elicit an outcome or an emotion in you? Indeed, how to achieve this is a problem that must be solved by the artist. Consider the painting *Sorrowing Old Man (At Eternity's Gate)* by Vincent van Gogh (search for it on the internet). Can you see the old man's sorrow? Can you imagine the weight of the years on him? If so, we can say that the painter successfully solved the problem. This brings us to another important component of problem solving: testing. It is not only important to solve a problem. It is equally important to test the solution to make sure it works and in many cases to make sure that the solution is efficient.

This book is about systematic problem solving or if you like about systematic reasoning. Unlike most textbooks about programming, this textbook is not about tinkering with or hacking code. This book is about making a plan to solve a problem and then implementing the solution. As we shall discover, it turns out that the solutions to many problems are similar. This should not come as a surprise because we often solve many problems using similar data. How do you do grocery shopping? You make a list of items and check them off as you put them in your cart. How do you manage your chores today? You make a list of chores and check them off as you get them done. Pretty similar, no? Similarities give rise to abstraction to avoid repetitions—or reinventing the proverbial wheel. This book, therefore, is also about abstraction. Thinking abstractly is a powerful tool in problem solving.

In this textbook, all the solutions to problems are expressed as programs. It is important to be somewhat precise about what a program is. A program is much more

than just code written using a programming language. Remember that a program is a solution to a problem. Therefore, a program has a design, code, examples of how it works, and tests. That is, it communicates how the problem is solved and illustrates that the solution works. If any of the mentioned components are missing, then we have an incomplete program. Would you believe someone who simply told you that n^2 , where n is a nonnegative integer, is the sum of the first n odd numbers? Many readers would be skeptical. What if they also provided the following examples:

$$0^2 = 0$$

$$2^2 = 1 + 3$$

$$4^2 = 1 + 3 + 5 + 7$$

It is very likely that most readers would now feel more confident that the claim is true. It is the same in programming. We cannot simply say that here is a function that does this or that. We need to explain how the function computes its value, and we need to have examples that show how it works. The steps taken to design a program in a systematic manner is called a *design recipe*. In this textbook, you shall study many different design recipes. Each design recipe shall become a tool in your problem-solving toolbox.

There are two problem-solving techniques that are emphasized throughout the book: *divide and conquer* and *iterative refinement*. Divide and conquer is the process by which a large problem is broken into two or more smaller problems that are easier to solve and then the solutions for the smaller pieces are combined to create an answer to the problem. Iterative refinement is the process by which a solution to a problem is gradually made better—like the drafts of an essay. Mastering these techniques is essential to becoming a good problem solver and programmer.

Finally, problem solving ought to be fun. To this end, this book promises that by the end of it you will have designed and implemented a multiplayer video game that you can play with your friends over the internet. To achieve this, however, there is a lot about problem solving and programming that you must first learn. The game is developed using iterative refinement. As we learn about programming, we shall apply our new knowledge to develop increasingly better versions of the video game. In fact, every skill you develop for problem solving and program design is transferable to other (non-programming) domains and to other programming languages.

1 The Languages and the Parts of the Book

The book uses the Racket student languages to write programs. These languages are chosen for several reasons. The first is that they have an error-messaging system specifically designed for beginners. This means that unlike common programming languages the error messages are likely to make sense to beginners. If you do not understand an error message, do not hesitate to ask your professor or search for help online. The second is that the syntax is simple and easy to understand. This is important because the emphasis is always on problem solving and not on how

to correctly write expressions. The third is that the student languages progressively become richer. At the beginning, you have fewer features at your disposal and, therefore, the possible errors are fewer. The fourth reason is that the student languages come with powerful libraries to create graphics, animations, and video games. These libraries allow students to inject their own personalities in the development of games and animations. You are strongly encouraged to be creative. Finally, the fifth reason is that the Racket student languages are likely to put all students on the same playing field. Most students will be learning the syntax of the programming language together for the first time.

The book is divided into five parts. Part I focuses on the basics. It starts with how to write expressions. Once expressions are mastered, the first abstraction lesson introduces us to functions. In addition, this part introduces you to conditional expressions that allow you to write programs that make decisions. Just this much knowledge allows us to write interactive programs and puts us on our way to a multiplayer video game. As you shall discover, decision-making is fundamental to solving problems that involve information that has many varieties. For example, the whole numbers may be positive or negative—two varieties—and how a whole number is processed depends on which variety a given number belongs to. Think about how to compute the absolute value of a whole number.

Part II introduces you to compound data of finite size. Compound data has multiple values associated. For example, a point on the Cartesian plane is compound data of finite size. There are two values: an x coordinate and a y coordinate. Being able to define compound data of finite size to represent elements in the real or an imaginary world is a powerful skill to develop.

Part III introduces you to compound data of arbitrary size. This is data that has multiple values, but the number of values is not fixed. Once again, think about a grocery list. Sometimes there are no items in the list and at other times there may be 10, 6, or 17 items in the list. This is where you are introduced to *structural recursion*—a powerful data-processing strategy that uses divide and conquer to process data whose size is not fixed. The types of data that are introduced are lists, intervals, natural numbers, and binary trees. The knowledge developed is used to develop a video game that is more challenging for the player.

Part IV delves into abstraction. This section is where we learn how to eliminate repetitions in our solutions to problems. In fact, we learn how different data can be processed and different problems can be solved in exactly the same way. You are introduced to *generic programming*, which is abstraction over the type of data processed. This leads to the realization that functions are data and, perhaps more surprising, that data are functions. In other words, the line between data and functions is artificial—a fact that is not emphasized enough in Computer Science textbooks. This realization naturally leads to object-oriented programming—a topic that you are likely to study extensively.

Part V introduces you to distributed programming—using multiple computers to solve a problem. This is a topic that until now has never been addressed in a textbook for beginning programmers. The fact that you develop proficiency in program design makes it possible for this topic, common in modern computer applications, to be

discussed. If you have ever sent a text message or have ever played a game online, then you have benefitted from and have used a distributed program. It is impossible, of course, to discuss all the nuances of distributed programming in this textbook. Nonetheless, you are introduced to a modern trend that is likely to be common throughout your professional career and beyond.

2 Acknowledgments

This book is the product of over ten years of work at Seton Hall University building on the shoulders of giants in Computer Science. There are many persons and groups who deserve credit for informing my work. The Racket community has been unequivocal in its support for the techniques that I have developed. There is an unpayable debt of gratitude owed to Matthias Felleisen from Northeastern University for our discussions over the years about Computer Science education, Liberal Arts education, and program design. My students and I have greatly benefitted from his support. Other Racketeers who have deeply influenced me are Shiram Krishnamurthi, Matthew Flatt, Robert Bruce Findler, and Kathi Fisler. This textbook is a tribute to our debates and their published work.

I would also like to thank the Trends in Functional Programming (TFP) and the Trends in Functional Programming in Education (TFPIE) communities. These communities provided (and continue to provide) a venue to discuss and present work advancing Computer Science education. I am grateful to many individuals including Peter Achten, Jurriaan Hage, Pieter Koopman, Simon Thompson, and Marko van Eekelen. Their insightful feedback has informed much of the material in this textbook.

Finally, I would like to thank Seton Hall University and its Department of Computer Science for supporting the development of the work presented in this textbook. In particular, the support of John T. Saccoman, Manfred Minimair, and Daniel Gross is appreciated. Most of all, I am grateful to all my CS1 students over the past decade who have informed my Computer Science education efforts. It is likely true that my students have learned a great deal in my courses, but it is an absolute certainty that I have learned more from them. They have refined the delivery of every idea found in this textbook. I am especially grateful to all my undergraduate tutors and teaching assistants, including Shamil Dzhatdoyev, Josie Des Rosiers, Nicholas Olson, Nicholas Nelson, Lindsey Reams, Craig Pelling, Barbara Mucha, Joshua Schappel, Sachin Mahashabde, Rositsa Abrasheva, Isabella Felix, and Sena Karsavran. Without my dedicated students at Seton Hall University and their insight into what students understood, this textbook would have been impossible.

Contents

Preface	vii
1 The Languages and the Parts of the Book	viii
2 Acknowledgments	x
 Part I The Basics of Problem Solving with a Computer	
1 The Science of Problem Solving	3
3 Getting Started	5
4 Computing New Values	8
5 Definitions and Interactions Areas Differences	11
6 Saving Your Work	12
7 Error Messages	12
7.1 Grammatical Errors	13
7.2 Type Errors	16
7.3 Runtime Errors	17
8 What Have We Learned in This Chapter?	19
2 Expressions and Data Types	21
9 Definitions	22
10 Numbers	25
11 Strings and Characters	27
12 Symbols	30
13 Booleans	31
13.1 Basic Boolean Operators in BSL	32
13.2 Predicates	34
14 Images	37
14.1 Basic Image Constructors	38
14.2 Property Selectors	38
14.3 Image Composers	41

14.4	Empty Scenes and Placing Images	42
15	What Have We Learned in This Chapter?	45
3	The Nature of Functions	47
16	The Rise of Functions	48
17	General Design Recipe for Functions	51
17.1	The Design Recipe in Action	53
18	Auxiliary Functions	55
18.1	Bottom-Up Design	56
19	Top-Down Design	61
20	What Have We Learned in This Chapter?	68
4	Aliens Attack Version 0	71
21	The Scene for Aliens Attack	72
22	Creating Aliens Attack Images	76
23	Shot Image	78
24	Alien Image	80
25	Rocket Image	82
25.1	Rocket Window Image Constructor	82
25.2	Rocket Fuselage Image Constructor	84
25.3	Rocket Single Booster Image Constructor	85
25.4	Rocket Booster Image Constructor	87
25.5	Rocket Main Body Image Constructor	88
25.6	Rocket Nacelle Image Constructor	90
25.7	Rocket ci Constructor	92
26	Drawing Functions	94
27	What Have We Learned in This Chapter?	100
5	Making Decisions	101
28	Conditional Expressions in BSL	102
29	Designing Functions to Process Data with Variety	104
30	Enumeration Types	110
31	Interval Types	118
32	Itemization Types	121
33	What Have We Learned in This Chapter?	125
6	Aliens Attack Version 1	127
34	The Universe Teachpack	128
35	A Video Game Design Recipe	139
36	Adding the Rocket to Aliens Attack	140
37	What Have We Learned in This Chapter?	149

Part II Compound Data of Finite Size

7	Structures	153
38	The posn Structure	155
39	Going Beyond the Design Recipe	160
40	Revisiting in-Q1?	162
41	What Have We Learned in This Chapter?	166
8	Defining Structures	167
42	Defining Structures	167
43	Computing Structures	171
44	Structures for the Masses	176
45	What Have We Learned in This Chapter?	183
9	Aliens Attack Version 2	185
46	Data Definitions	186
47	Function Templates and Sample Instances	187
48	The run Function	190
49	Drawing the World	191
49.1	The draw-world Refinement	191
49.2	Drawing Aliens	193
50	The process-key Refinement	195
51	Processing Ticks	197
51.1	The process-tick Handler	197
51.2	The Design of new-dir-after-tick	199
51.3	The Design of Auxiliary Functions for new-dir-after-tick	202
51.4	The Design of move-alien	206
52	Subtyping	209
52.1	Checking Errors	214
53	The game-over? Handler	218
54	Computing the Last Scene	220
55	What Have We Learned in This Chapter?	223
10	Structures and Variety	225
56	A Bottom-Up Design	226
57	Code Refactoring	233
58	What Have We Learned in This Chapter?	236
11	Aliens Attack Version 3	239
59	Data Definitions	240
60	The draw-world Refinement	243
61	The process-key Refinement	246
62	The process-tick Refinement	251
62.1	The Refinement	251
62.2	The move-shot Design	253
63	The game-over? Refinement	256
63.1	The hit? Design	257

63.2	The draw-last-world Refinement	259
64	What Have We Learned in This Chapter?	261

Part III Compound Data of Arbitrary Size

12	Lists	265
65	Creating and Accessing Lists in ISL+	266
66	Shorthand for Building Lists	269
67	Recursive Data Definitions	271
68	Generic Data Definitions	274
69	Function Templates for Lists	275
70	Designing List-Processing Functions	277
71	What Have We Learned in This Chapter?	279
13	List Processing	281
72	List Summarizing	281
73	List Searching	286
74	List ORing	288
74.1	Determining If an Alien Is at the Left Edge	288
74.2	Determining If an Alien Is at the Right Edge	291
74.3	Determining If an Alien Has Reached Earth	292
75	List ANDing	294
75.1	All Even in a 1on	295
75.2	Determining if a 1on Is Sorted	296
76	List Mapping	300
76.1	Moving a List of Aliens	300
76.2	Moving a List of Shots	301
76.3	Returning a Different List Type	302
77	List Filtering	304
77.1	Extracting Even numbers	305
77.2	Removing Hit Aliens	306
77.3	Removing Shots	310
78	List Sorting	314
79	What Have We Learned in This Chapter?	318
14	Natural Numbers	319
80	Data Definition for a Natural Number	320
81	Computing Factorial	321
82	Computing Tetrahedral Numbers	323
83	Making Copies	327
84	What Have We Learned in This Chapter?	329

15	Interval Processing	331
85	Interval Data Definition	332
86	Revisiting Factorial	334
87	Creating an Army of Aliens	336
88	Largest Prime in an Interval	342
89	What Have We Learned in This Chapter?	347
16	Aliens Attack Version 4	349
90	New world Data Definition and Function Template	349
91	The draw-world Refinement	351
92	The process-key Refinement	354
93	The process-tick Refinement	358
93.1	The new-dir-after-tick Design	361
94	The game-over? Refinement	365
94.1	The draw-last-world Refinement	367
95	A Bug Despite Hundreds of Tests Passing	369
96	What Have We Learned in This Chapter?	372
17	Binary Trees	373
97	Binary Tree Data Definition	374
98	Traversing a Binary Tree	376
99	The Maximum of a (btof int)	379
100	Binary Search Trees	382
100.1	A (listof cr) Representation	383
100.2	A (btof cr) Representation	384
100.3	A (bstof cr) Representation	386
101	Abstract Running Time	388
102	The Complexity of Searching the Criminal Database	391
103	Balanced (bstof cr)	393
103.1	Creating a Balanced Binary Search Tree	393
103.2	Analysis	396
104	What Have We Learned in This Chapter?	398
18	Mutually Recursive Data	401
105	Designing with Mutually Recursive Data	403
105.1	Revisiting the Maximum of a (btof int)	403
106	Evaluating Arithmetic Expressions	407
107	Trees	414
107.1	Creating a Search Tree for Tic Tac Toe	418
107.2	Can Win Tic Tac Toe?	422
108	Project: Tic Tac Toe	426
108.1	Data Analysis	427
108.2	Design draw-world	428
108.3	Design process-mouse	428
108.4	Design process-tick	429

108.5	Design game-over?	430
109	What Have We Learned in This Chapter?	430
19	Processing Multiple Inputs of Arbitrary Size	433
110	One Input Has a Dominant Role	433
111	Inputs Must Be Processed Simultaneously	436
112	No Clear Relationship Between the Inputs	438
113	What Have We Learned in This Chapter?	442
 Part IV Abstraction		
20	Functional Abstraction	445
114	A Design Recipe for Abstraction	447
115	Functions as Values	447
116	Abstraction Over List-Processing Functions	450
116.1	List Summarizing	450
116.2	List Searching	456
116.3	List ORing	459
116.4	List ANDing	462
116.5	List Mapping	464
116.6	List Filtering	466
116.7	List Sorting	467
117	Abstraction over Interval-Processing Functions	472
118	What Have We Learned in This Chapter?	475
21	Encapsulation	477
119	Local-Expressions	477
120	Lexical Scoping	480
121	Using Local-Expressions	483
121.1	Encapsulation	483
121.2	Readability	488
121.3	Furthering Functional Abstraction	490
121.4	One-Time Expression Evaluation	492
122	What Have We Learned in This Chapter?	497
22	Lambda Expressions	499
123	Anonymous Functions	501
124	Revisiting Function Composition	503
125	Curried Functions	505
126	Designing Using Existing Abstractions	511
126.1	Computing the Value of a Series	511
126.2	Approximating π	514
127	What Have We Learned in This Chapter?	516

23	Aliens Attack Version 5	517
128	Constants	518
129	Structure Definitions	522
130	Encapsulating and Refactoring Handlers	522
130.1	The draw-world Handler	522
130.2	The process-key Handler	525
130.3	The process-tick Handler	526
130.4	The game-over? Handler	532
130.5	The draw-last-world Handler	533
131	Refactoring run	534
132	What Have We Learned in This Chapter?	535
24	For-Loops and Pattern Matching	537
133	For-Loops	538
133.1	for-loops	538
133.2	for*-loops	542
134	Pattern Matching	546
134.1	Illustrative Example	547
134.2	Refactoring Using Pattern Matching	549
134.3	Designing Using Pattern Matching	551
135	What Have We Learned in This Chapter?	554
25	Interfaces and Objects	557
136	Interfaces	558
136.1	Improving the Human Interface	561
136.2	Services that Require More Input	561
137	A Design Recipe for Interfaces	565
138	Interfaces and Union Types	566
139	An Abbreviated (listof X) Interface	567
139.1	Step 1: Values and Services	567
139.2	Step 2: Interface and Message Definitions	568
139.3	Step 3: Class Function Template	568
140	The Empty (listof X) Class	571
140.1	Step 4: Signature, Purpose, Class Header, and Message-Passing Function	571
140.2	Step 5: Auxiliary Functions	572
141	The Non-Empty (listof X) Class	573
141.1	Step 4: Signature, Purpose, Class Header, and Message-Passing Function	573
141.2	Step 5: Auxiliary Functions	574
142	Step 6: Wrapper Functions and Tests	575
143	What Have We Learned in This Chapter?	579

Part V Distributed Programming

26	Introduction to Distributed Programming	583
144	A Design Recipe for Distributed Programming	585
145	More on the Universe API	586
146	A Chat Application	589
146.1	The Components	589
146.2	Data Definitions	589
146.3	Communication Protocol	592
146.4	Marshalling and Unmarshalling	593
146.5	Component Implementation	593
146.6	Running the Chat Tool	600
147	What Have We Learned in This Chapter?	601
27	Aliens Attack Version 6	603
148	Refining the world Data Definition	603
149	The draw-world Refinement	607
150	The process-key Refinement	610
151	The process-tick Refinement	615
152	The game-over? Refinement	617
153	What Have We Learned in This Chapter?	619
28	Aliens Attack Version 7	621
154	Components	621
155	Data Definitions	622
156	Communication Protocol	623
156.1	Player-Sparked Communication Chains	623
156.2	Server-Sparked Communication Chains	624
156.3	Message Data Definitions	626
157	Marshalling and Unmarshalling	632
158	Component Implementation	638
158.1	Player Component	638
158.2	Server Component	646
159	A Subtle Bug	653
160	What Have We Learned in This Chapter?	656
29	Aliens Attack Version 8	657
161	The Components	657
162	Data Definitions	658
163	Communication Protocol	659
163.1	Player-Sparked Communication Chains	660
163.2	Server-Sparked Communication Chains	661
163.3	Message Data Definitions	662
164	Marshalling and Unmarshalling	664
165	Component Implementation	665
165.1	Player Component	666
165.2	Server Component	669

166 A Subtle Problem 683
 167 What Have We Learned in This Chapter? 684

Part VI Epilogue

30 Advice for Future Steps 687
 168 Advice for Computer Science Students 687
 169 Advice for Non-Computer Science Students 688