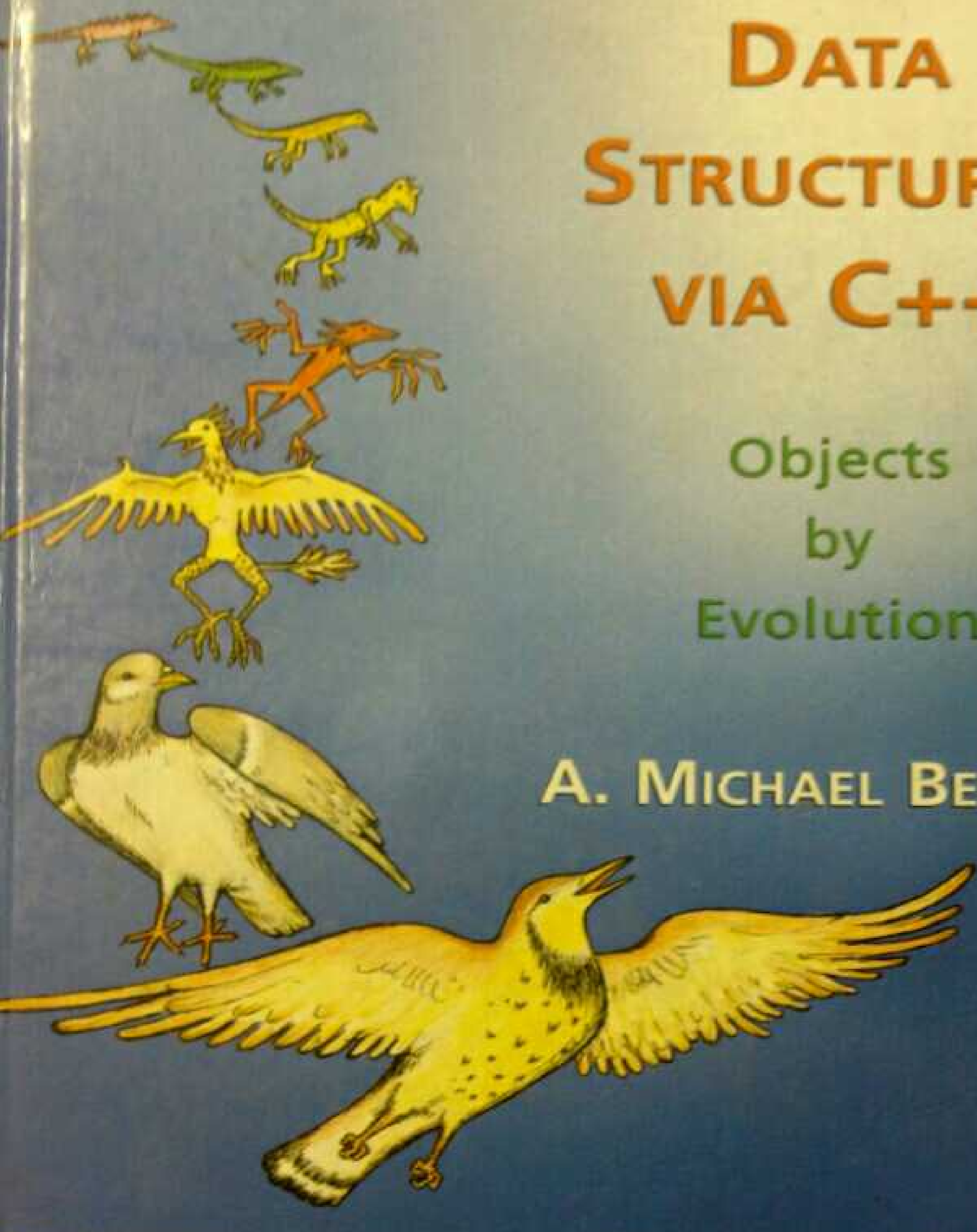
# DATA STRUCTURES VIA C++

Objects by Evolution

A. MICHAEL BERMAN



# Data Structures via C++

## Objects by Evolution

A. Michael Berman
*Rowan College of New Jersey*

1162
SEMINAR LIBRARY
Department of Computer Science
UNIVERSITY OF KARACHI
17.10.98

New York Oxford
OXFORD UNIVERSITY PRESS
1997

# Contents

Preface xiii

Chapter 1 **Software Engineering and Computer Programming** 1

1.1 Software Engineering and Computer Science 2

1.1.1 Data Structures and Abstract Data Types 4

1.2 The Software "Life Cycle" 9

1.2.1 The Waterfall 9

1.2.2 Critiques of the Waterfall 12

1.2.3 Documentation 13

1.3 Why C++? 14

Chapter 2 **Designing Software: Two Approaches** 19

2.1 Why Design? 20

2.2 Top-Down Design 21

2.3 The Object Alternative 25

2.4 Which Method Is Better, TDD or OOD? 31

Chapter 3 **Software Reliability** 37

3.1 Risks of Faulty Software 38

3.2 Testing 40

3.2.1 A Taxonomy of Errors 40

3.2.2 Approaches to Testing 44

3.2.3 Two Techniques for Unit Testing 47

3.3 Applying Program Correctness Techniques 50
3.3.1 Assertions 50
3.3.2 Preconditions and Postconditions 52
3.3.3 Loop Invariants and Proving Termination 53
3.3.4 Insertion Sort 57

Chapter 4 Abstract Data Types, Classes, and Objects 67
4.1 Problem: Computing with Time 68
4.2 Describing Data Types 69
4.2.1 What's an ADT and Why Use It? 69
4.2.2 A Time ADT 71
4.3 ADT Implementation and Code Reuse 74
4.3.1 Code Reuse: Don't Reinvent the Wheel 74
4.3.2 Implementing a Time ADT 75
4.4 Information Hiding, Encapsulation, and Views 77
4.5 Creating Encapsulated ADTs Using the C++ Class 78
4.5.1 Declaring the Time Class 79
4.5.2 Creating ADTs for C++ Classes 81
4.5.3 Creating Clients for C++ Classes 83
4.5.4 Implementation for C++ Classes 84
4.6 Using Standard C++ Class Libraries 85
4.6.1 Using C++ Libraries for Input and Output 86
4.6.2 Using the C++ String Library 89
4.7 ADTs, Objects, and Object-Oriented Programming 92

Chapter 5 Efficiency 99
5.1 Selecting Good Algorithms 100
5.2 The Many Faces of Program Efficiency 101
5.3 Algorithms for Searching 103
5.3.1 Linear Search 104
5.3.2 Binary Search 108
5.4 Analysis of Some Simple Sorting Algorithms 114
5.4.1 Selection Sort 114
5.4.2 Bubble Sort 119

Chapter 6 Recursion 127
6.1 Solving Problems with Recursion 128
6.1.1 A Very Simple Example 128
6.1.2 The Nature of Recursion 130
6.2 Recursive Definitions 130
6.3 Applying Recursion to Sorting and Searching Problems 135
6.3.1 Recursive Search Algorithms 135
6.3.2 Quicksort: A Recursive Sorting Algorithm 139

6.4 How Is Recursion Implemented? 151
- 6.4.1 What Happens When a Function Is Called? 151
- 6.4.2 What Happens When a Recursive Function Is Called? 152
- 6.4.3 Is Recursion Inefficient? 153

Chapter 7 **Lists** 161

7.1 Problem: A Membership Management Program 162
7.2 The List ADT 162
7.3 Implementing Lists 164
- 7.3.1 A Header File for the List ADT 164
- 7.3.2 Implementation via Arrays 166
- 7.3.3 Implementation via Linked Lists 170
- 7.3.4 Dynamic Memory Allocation 179

7.4 The Inorder List ADT 183
7.5 Variations on a Linked List 190
- 7.5.1 Dummy Head Nodes 190
- 7.5.2 Circular Linked Lists 196
- 7.5.3 Doubly Linked Lists 197

7.6 A Dynamic Linear List 200
7.7 The Membership Management Program Revisited 202

Chapter 8 **Stacks** 211

8.1 Problem: Robot Navigation 212
8.2 The Stack ADT 215
8.3 Implementing the Stack 1: Array 216
8.4 Creating Generic Classes with Templates 221
8.5 Implementing the Stack 2: Dynamic List 225
8.6 Applications of the Stack ADT 227
- 8.6.1 Building a Calculator with a Stack 227
- 8.6.2 How Is Recursion Implemented? Part 2 232

8.7 The Robot Navigation Problem Solved 237

Chapter 9 **Queues** 251

9.1 Problem: Computer Network Performance 250
9.2 The Queue ADT 254
9.3 Implementing a Queue 1: Array 256
9.4 Implementing a Queue 2: Dynamic List 262
9.5 Simulation: Modelling a Computer Network 265

Chapter 10 **Tables** 279

10.1 A Data Structure to Support Retrieval by Key 280
- 10.1.1 The Routing Problem 280
- 10.1.2 Defining the Table ADT 281

10.2 Implementing a Table 282
- 10.2.1 A Simple Implementation That Doesn't Quite Work 282

10.3 Hash Tables for Fast Retrieval 285
10.3.1 Hash Functions 285
10.3.2 Picking a Good Hash Function 287
10.3.3 Methods of Handling Collisions 288
10.3.4 A Complete Implementation 292
10.3.5 Chained Hashing 297
10.4 Using Tables 302

Chapter 11 Trees 307
11.1 Introducing Trees 308
11.1.1 Talking About Trees 308
11.1.2 Binary Trees 309
11.1.3 An Application: Expression Trees 311
11.2 Building the Binary Tree 314
11.2.1 The Binary Tree ADT 314
11.2.2 Implementing a Binary Tree 315
11.2.3 A Sample Client for the Binary Tree 318
11.2.4 Using the Binary Tree: Expression Trees Revisited 321
11.3 Tree Traversal 325
11.3.1 Three Tree Traversal Algorithms 325
11.3.2 Using Tree Traversals 329
11.3.3 Implementing Tree Traversals 330
11.4 Binary Search Trees 336
11.4.1 An Ordered Tree ADT 336
11.4.2 Implementing the Binary Search Tree 340
11.5 Reuse Through Inheritance: A Hierarchy of Trees 345
11.6 Performance of Binary Trees 353
11.6.1 The Shapes of Binary Trees 353

Chapter 12 Graphs 361
12.1 Example: Keeping Track of Course Prerequisites 362
12.2 Basic Graph Concepts and Terminology 363
12.3 Creating Graph ADTs 365
12.3.1 Data Structures to Model Graphs 365
12.3.2 Defining Graph ADTs 368
12.3.3 A Hierarchy of Graphs 374
12.4 Implementing and Using Adjacency List Graphs 376
12.4.1 Implementing Lists with Iterators 376
12.4.2 The Adjacency List Abstract Base Class 380
12.4.3 The Undirected and Directed Adjacency List Graphs 381
12.4.4 Topological Sort 384
12.5 Implementing and Using Adjacency Matrix Graphs 390
12.5.1 Implementation of the Adjacency Matrix Classes 390
12.5.2 Finding the Transitive Closure 393