

2 Records, Arrays, and Pointers 48

- 2.1 Bricks 48
 - 2.1.1 Pascal Records 49
 - 2.1.2 Arrays in Pascal 50
 - 2.1.3 Representation of Records in Arrays 56
 - 2.1.4 Variable-Length Records 60
 - 2.1.5 Variant Records 62
 - 2.1.6 Address Calculation without Access 62
- 2.2 Mortar 63
 - 2.2.1 Pointers 63
 - 2.2.2 Pointer Variables and the Pascal Heap 65
- 2.3 Representations of Two-Dimensional Arrays 67
 - 2.3.1 Rowwise and Columnwise Representation 68
 - 2.3.2 Symmetric Array Representation 70
 - 2.3.3 Pointer Array Representation 71
- 2.4 Advantages and Disadvantages of the Techniques 73
- 2.5 Case Study: The Stable Marriage Problem 74
 - 2.5.1 The Algorithm 75
 - 2.5.2 The Program 78
- Exercises 81
- Suggested Assignment 83

3 Lists 86

- 3.1 Why Are Lists Needed? 86
- 3.2 Keeping Track of List Pointers 88
 - 3.2.1 Insertion and Deletion of Records in Lists 90
- 3.3 Expanding and Contracting Lists 91
 - 3.3.1 Insertion 91
 - 3.3.2 Special Cases 92
 - 3.3.3 Header Records 93
 - 3.3.4 Deletion 93
- 3.4 Traversal of Lists 95
 - 3.4.1 List Reversal Using a Loop 95
 - 3.4.2 A General List Traversal Using a Loop 98
 - 3.4.3 The Merits of Modularization 99
- 3.5 Using the TRAVERSE Procedure for Lists 99
- 3.6 Implementing Lists 108
 - 3.6.1 Lists Stored in the Pascal Heap 108
 - 3.6.2 Lists Stored in an Array of Records 109
 - 3.6.3 Lists Stored in Languages without Records 110
- 3.7 Keeping Track of Available Records 111
 - 3.7.1 Why Heap? (Using Dynamic Storage Management) 112
 - 3.7.2 How to Heap 113
- 3.8 Sequential Arrays versus Lists for Record Storage 116
- 3.9 Case Study: Merging and the Perfect Shuffle 117
 - 3.9.1 The Perfect Shuffle 117

3.9.2 Array Implementation of the Perfect Shuffle 118

3.9.3 List Implementation of the Perfect Shuffle 120

3.9.4 Merging Sequences of Entries 124

Exercises 126

Suggested Assignment 129

4 Introduction to Recursion, Stacks, and Queues 130

4.1 What Is Recursion? 130

4.2 Using Recursion 131

4.2.1 The Towers of Hanoi 131

4.2.2 Verifying and Simulating a Recursive Program 134

4.2.3 The Length of a List 137

4.2.4 Copying a List 138

4.2.5 **var** versus Non-**var** Parameters 139

4.2.6 Counting Squares 142

4.2.7 Permutations 143

4.3 A Close Look at the Execution of Recursive Programs 150

4.4 Implementing Recursive Programs 153

4.4.1 A Sample Implementation of TOWERS 154

4.4.2 A Sample Implementation of PERMUTATIONS 158

4.5 Stacks 161

4.5.1 Array Implementation of a Stack 162

4.5.2 List Implementation of the Stack with an Array of
Records 164

4.5.3 List Implementation of the Stack with Records in the Pascal
Heap 165

4.6 Case Study: Checking Sequences for Proper Nesting 166

4.7 Queues 172

4.7.1 Array and Circular Array Implementation of the Queue 172

4.7.2 List Implementation of the Queue with an Array of
Records 174

4.7.3 List Implementation of the Queue with Records in the Pascal
Heap 174

Exercises 175

Suggested Assignment 178

5 More Complex Lists 179

5.1 Imposing List-structure on Data 179

5.2 Traversal of List-structures 183

5.2.1 An Iterative Procedure 185

5.2.2 A Recursive Procedure 187

5.3 Using the TRAVERSE Procedures for List-structures 189

5.4 Implementing List-structures 194

5.5 Case Study: Information Retrieval 195

5.5.1 Family Relationships 196

5.5.2	A First Solution	196
5.5.3	A Better Solution	197
5.5.4	Updating the System	198
5.5.5	Retrieving Information from the System	204
5.5.6	Other System Components	205
	Exercises	205
	Suggested Assignment	208

6 Trees 211

6.1	Branching Out	211
6.1.1	Depth versus Capacity	215
6.2	Trees of Records	216
6.2.1	Insertion and Deletion of Records in Binary Trees	216
6.2.2	Climbing Binary Trees	218
6.2.3	Three Preorder Traversals of a Binary Tree	220
6.3	Using the Traverse Procedures for Binary Trees	223
6.4	Implementing Binary Trees	226
6.4.1	Sequential Representation	226
6.4.2	Linked Representation	227
6.4.3	List-structure Representation	229
6.5	Trees	229
6.5.1	Implementing Trees as Binary Trees	229
6.6	Traversals of Trees	232
6.6.1	Obtaining the Binary Tree for a Tree	233
6.6.2	Backtracking: The <i>N</i> -queens Problem	237
6.6.3	Depth-First	240
6.6.4	Breadth-First	245
6.6.5	Branch and Bound	245
6.7	More on Recursion, Trees, and Stacks	246
6.7.1	Balanced Binary Trees	246
6.7.2	Trading Storage for Time	250
	Exercises	252
	Suggested Assignment	257

APPLICATIONS

7 Searching and Sorting 258

7.1	Overview	258
7.2	Elementary Searches	259
7.2.1	Linear Search	260
7.2.2	Saving Time: A Neat Trick	260
7.2.3	Binary Search	262

7.2.4	Timing the Binary Search	264
7.2.5	Interpolation Search	264
7.3	Elementary Sorts	265
7.3.1	Maximum Entry Sort	265
7.3.2	Bubble Sort	267
7.3.3	Timing the Bubble Sort	268
7.3.4	Insertion Sort	269
7.3.5	Timing the Insertion Sort	270
7.3.6	Attempted Improvements	271
7.4	Heapsort: A Faster Sort	272
7.4.1	Heapsorts	273
7.4.2	Creating a Heap	275
7.4.3	Timing the Heap Creation	275
7.4.4	Better Heap Creation	277
7.4.5	Some Details of Heap Implementation	280
7.4.6	Reheaping	280
7.4.7	An Implementation for Heapsort	281
7.5	Quicksort: Another Fast Sort	283
7.5.1	Two Quicksort Procedures	285
7.5.2	The Partition Module	287
7.5.3	Analyzing Iterative Quicksort	290
7.5.4	The Worst and Best Cases	291
7.5.5	Distributive Partitioning	292
7.6	Priority Queues	292
7.6.1	Simple Implementations	293
7.6.2	Using a Heap	293
7.6.3	Using Leftist Trees	294
7.7	Binary Search Trees	296
7.7.1	Searching the Search Tree	297
7.7.2	Growing the Search Tree “Simply”	298
7.7.3	The Shape of Simple Binary Search Trees	299
7.7.4	Deleting a Record “Simply”	300
7.7.5	A Balancing Act	300
7.7.6	Maintaining Balance	308
7.7.7	Random Access in AVL Trees	317
7.8	Hash Tables	318
7.8.1	Building a Hash Table	319
7.8.2	Searching a Hash Table	320
7.8.3	Random Hashing	322
7.8.4	Other Collision Resolution Policies	323
7.8.5	Searching in Ordered Hash Tables	331
7.8.6	Deletion from Hash Tables	331
7.9	Simulation of an Algorithm	332
7.10	Synopsis of Search and Sort Efficiencies	335
	Exercises	337
	Suggested Assignment	340

8 Files 341

- 8.1 The File Data Structure 341
- 8.2 Internal and External Memory 342
 - 8.2.1 Operating Systems 342
 - 8.2.2 Filters 342
- 8.3 Organization of Files 344
 - 8.3.1 Sequential Files 345
 - 8.3.2 Text Files 351
- 8.4 Sequential Access in External Memory 352
- 8.5 Sorting Tape Files 354
 - 8.5.1 Straight Merge 354
 - 8.5.2 Natural Merge 356
 - 8.5.3 Replacement Selection 357
 - 8.5.4 Polyphase Sort 358
- 8.6 Direct Access in External Memory 360
- 8.7 Sequential and Random Access of Disk Files 362
 - 8.7.1 Sequential Access 362
 - 8.7.2 Random Access 364
 - 8.7.3 Indexed Sequential Access 364
 - 8.7.4 B-trees for Sequential and Random Access 366
- 8.8 Summary 372
- Exercises 373
- Suggested Assignment 374

9 Topological Sorting: An Archetypal Solution 376

- 9.1 Background
 - 9.1.1 Binary Relations and Partial Orders 377
 - 9.1.2 Graphic Representation of Partial Orders 378
 - 9.1.3 Topological Sorts: Consistent Rankings 379
- 9.2 A Searching Solution 381
- 9.3 A Constructed Solution 381
 - 9.3.1 Correctness 382
 - 9.3.2 An Initial Implementation 384
 - 9.3.3 A Better Implementation 385
- 9.4 Analysis of TOPSORT 395
- 9.5 Behavior for Replicated Pairs or Loops 396
- 9.6 Final Comments on TOPSORT 397
 - 9.6.1 An Input Validation Module 398
- 9.7 Reviewing Methodology 398
- Exercises 398
- Suggested Assignment 400

10 Compilers 402

- 10.1 How Compilers Work 402

10.2	Language Specification	403
10.2.1	Derivations and Derivation Trees	404
10.2.2	Syntax and Semantics	406
10.2.3	Parsing	408
10.2.4	Lexical Analysis and Symbol Tables	412
10.2.5	Translation	414
10.3	Evaluation and Translation of Expressions	415
10.3.1	Postfix Expressions	416
10.3.2	Infix Expressions	418
10.3.3	Translating Infix to Postfix	422
10.4	Top-down and Bottom-up Compilers	422
10.5	A Simple Compiler	424
10.6	Compiler Conclusions	431
	Exercises	431
	Suggested Assignment	432

11 Huffman Coding and Optimal and Nearly Optimal Binary Search Trees 433

11.1	Techniques for Compressing Text or Storing Records	433
11.2	Weighted Path Length	434
11.3	Huffman Coding	435
11.3.1	The Huffman Algorithm	438
11.3.2	Representation of Huffman Trees	440
11.3.3	Implementation	440
11.3.4	A Proof	442
11.4	Optimal Binary Search Trees	444
11.4.1	Finding Optimal Binary Search Trees	445
11.5	Nearly Optimal Binary Search Trees	449
11.5.1	Greedy Binary Search Trees	450
11.5.2	Greedy Construction	452
11.5.3	Implementation	454
11.5.4	Alphabetic Codes	455
11.6	Conclusion	455
	Exercises	457
	Suggested Assignment	459

12 Some Pointers on Storage Management 460

12.1	The Need for Storage Management	460
12.2	The Heap	462
12.3	The List of Available Space for Variable-Length Records	463
12.3.1	Two-Way Circular Lists	465
12.3.2	The Buddy System	466
12.3.3	Indexing Methods	468
12.4	Shared Storage	469

12.5 Storage Reclamation Techniques	470
12.5.1 Garbage Collection	470
12.5.2 The Reference Counter Method	471
12.6 Stackless Traversals	475
12.6.1 Threaded Trees	477
12.6.2 Link Inversion	478
12.6.3 Robson Traversal	480
12.7 Pitfalls: Garbage Generation and Dangling References	483
12.8 Conclusion	486
Exercises	486
Suggested Assignment	488

Bibliography 489**Index 494**

