Contents

PART ONE
Preliminaries 1

1 An Overview of ANSI C **3**

1.1 **What is C? 4**

1.2 **The structure of a C program 5**
Comments 7, Library inclusions 8, Program-level definitions 8, Function prototypes 9, The main program 9, Function definitions 10

1.3 **Variables, values, and types 11**
Variables 11, Naming conventions 12, Local and global variables 13, The concept of a data type 13, Integer types 14, Floating-point types 15, Text types 16, Boolean type 18, Simple input and output 18

1.4 **Expressions 20**
Precedence and associativity 21, Mixing types in an expression 22, Integer division and the remainder operator 23, Type casts 24, The assignment operator 24, Increment and decrement operators 26, Boolean operators 28

1.5 **Statements 30**
Simple statements 30, Blocks 30, The **if** statement 31, The **switch** statement 32, The **while** statement 34, The **for** statement 36

1.6 **Functions 39**
Returning results from functions 39, Function definitions and prototypes 40, The mechanics of the function-calling process 40, Stepwise refinement 41

Summary 42
Review questions 43
Programming exercises 45

2 Data Types in C **51**

2.1 **Enumeration types 52**
Internal representation of enumeration types 53, Scalar types 54, Understanding **typedef** 55

2.2 **Data and memory 56**
Bits, bytes, and words 56, Memory addresses 57

2.3 **Pointers 59**
Using addresses as data values 60, Declaring pointer variables 60, The fundamental pointer operations 61, The special pointer **NULL** 64, Passing parameters by reference 64

2.4 **Arrays 66**
Array declaration 69, Array selection 70, Effective and allocated sizes 71, Passing arrays as parameters 72, Initialization of arrays 72, Multidimensional arrays 75

2.5 **Pointers and arrays 77**
Pointer arithmetic 78, Incrementing and decrementing pointers 81, The relationship between pointers and arrays 82

2.6 **Records 84**
Defining a new structure type 85, Declaring structure variables 85, Record selection 86, Initializing records 86, Pointers to records 87

2.7 **Dynamic allocation 88**
The type `void *` 89, Coping with memory limitations 90, Dynamic arrays 91, Dynamic records 93

Summary 94

Review questions 95

Programming exercises 98

3 Libraries and Interfaces 107

3.1 **The concept of an interface 108**
Interfaces and implementations 108, Packages and abstractions 109, Principles of good interface design 110

3.2 **Random numbers 111**
The structure of the `random.h` interface 111, Constructing a client program 115, The ANSI functions for random numbers 117, The `random.c` implementation 120

3.3 **Strings 123**
The underlying representation of a string 124, The data type `string` 125, The ANSI string library 127, The `strlib.h` interface 132

3.4 **The standard I/O library 138**
Data files 138, Using files in C 139, Standard files 141, Character I/O 141, Rereading characters from an input file 142, Updating a file 142, Line-oriented I/O 145, Formatted I/O 146, The `scanf` functions 146

3.5 **Other ANSI libraries 148**

Summary 150

Review questions 151

Programming exercises 154

PART TWO
Recursion and Algorithmic Analysis **161**

4 Introduction to Recursion 163

4.1 A simple example of recursion 164
4.2 The factorial function 166
The recursive formulation of `Fact` 167, Tracing the recursive process 167, The recursive leap of faith 171
4.3 The Fibonacci function 172
Computing terms in the Fibonacci sequence 173, Gaining confidence in the recursive implementation 174, Efficiency of the recursive implementation 176, Recursion is not to blame 176
4.4 Other examples of recursion 178
Detecting palindromes 179, Binary search 180, Mutual recursion 182
4.5 Thinking recursively 185
Maintaining a holistic perspective 185, Avoiding the common pitfalls 186
Summary 187
Review questions 188
Programming exercises 190

5 Recursive Procedures 195

5.1 The Tower of Hanoi 196
Framing the problem 197, Finding a recursive strategy 198, Validating the strategy 200, Coding the solution 201, Tracing the recursive process 201
5.2 Generating permutations 206
The recursive insight 207
5.3 Graphical applications of recursion 208
The graphics library 209, An example from computer art 212, Fractals 217
Summary 222
Review questions 223
Programming exercises 224

6 Backtracking Algorithms 235

6.1 Solving a maze by recursive backtracking 236
The right-hand rule 236, Finding a recursive approach 237, Identifying the simple cases 238, Coding the maze solution algorithm 239, Convincing yourself that the solution works 243

6.2 **Backtracking and games 245**
The game of nim 246, A generalized program for two-player games 248, The minimax strategy 254, Implementing the minimax algorithm 257, Using the general strategy to solve a specific game 259
Summary 272
Review questions 272
Programming exercises 274

7 Algorithmic Analysis 283

7.1 **The sorting problem 284**
The selection sort algorithm 285, Empirical measurements of performance 286, Analyzing the performance of selection sort 287

7.2 **Computational complexity 288**
Big-O notation 289, Standard simplifications of big-O 290, The computational complexity of selection sort 290, Predicting computational complexity from code structure 291, Worst-case versus average-case complexity 293, A formal definition of big-O 294

7.3 **Recursion to the rescue 296**
The power of divide-and-conquer strategies 296, Merging two arrays 297, The merge sort algorithm 298, The computational complexity of merge sort 300, Comparing N^2 and $N \log N$ performance 302

7.4 **Standard complexity classes 303**

7.5 **The Quicksort algorithm 306**
Partitioning the array 308, Analyzing the performance of Quicksort 311

7.6 **Mathematical induction 312**
Summary 315
Review questions 316
Programming exercises 318

PART THREE
***Data Abstraction* 325**

8 Abstract Data Types 327

8.1 **Stacks 328**
The basic stack metaphor 329, Stacks and function calls 329, Stacks and pocket calculators 330

8.2 **Defining a stack ADT 331**
Defining the types for the stack abstraction 331, Opaque types 333, Defining the **`stack.h`** interface 334

8.3 **Using stacks in an application 338**

8.4 **Implementing the stack abstraction 342**
Defining the concrete type 342, Implementing the stack operations 342, The advantages of opaque types 344, Improving the **`stack.c`** implementation 345

8.5 **Defining a scanner ADT 347**
The dangers of encapsulated state 347, Abstract data types as an alternative to encapsulated state 348, Implementing the scanner abstraction 353

Summary 358

Review questions 359

Programming exercises 360

9 Efficiency and ADTs 373

9.1 **The concept of an editor buffer 374**

9.2 **Defining the buffer abstraction 375**
Functions in the **`buffer.h`** interface 376, Coding the editor application 379

9.3 **Implementing the editor using arrays 380**
Defining the concrete type 381, Implementing the buffer operations 382, The computational complexity of the array implementation 385

9.4 **Implementing the editor using stacks 386**
Defining the concrete structure for the stack-based buffer 387, Implementing the buffer operations 387, Comparing computational complexities 388

9.5 **Implementing the editor using linked lists 391**
The concept of a linked list 392, Designing a linked-list data structure 393, Using a linked list to represent the buffer 394, Insertion into a linked-list buffer 396, Deletion in a linked-list buffer 398, Cursor motion in the linked-list representation 399, Linked-list idioms 402, Completing the buffer implementation 403, Computational complexity of the linked-list buffer 404, Doubly linked lists 407, Time-space tradeoffs 408

Summary 409

Review questions 410

Programming exercises 411

10 Linear Structures **419**

10.1 **Stacks revisited 420**

10.2 **Queues 424**
The structure of the `queue.h` interface 427, Array-based implementation of queues 427, Linked-list representation of queues 433

10.3 **Simulations involving queues 436**
Simulations and models 439, The waiting-line model 440, Discrete time 440, Events in simulated time 441, Implementing the simulation 442

Summary 448

Review questions 449

Programming exercises 451

11 Symbol Tables **457**

11.1 **Defining a symbol table abstraction 458**
Choosing types for values and keys 458, Representing an undefined entry 460, A preliminary version of the symbol table interface 461

11.2 **Hashing 462**
Implementing the hash table strategy 463, Choosing a hash function 468, Determining the number of buckets 470

11.3 **Limitations of the preliminary interface 471**

11.4 **Using functions as data 473**
A general plotting function 473, Declaring pointers to functions and function classes 474, Implementing `PlotFunction` 475, The `qsort` function 476

11.5 **Mapping functions 481**
Mapping over entries in a symbol table 481, Implementing `MapSymbolTable` 484, Passing client data to callback functions 485

11.6 **Iterators 486**
Using iterators 487, Defining the iterator interface 488, Implementing the iterator abstraction for symbol tables 488

11.7 **Command dispatch tables 492**

Summary 496

Review questions 497

Programming exercises 499

PART FOUR
Recursive Data **505**

12 Recursive Lists **507**

12.1 **The recursive formulation of a list 508**

12.2 **Defining an abstract list type 510**
Immutable types 513, Functions that manipulate list structure 514, Concatenating lists 517, Internal sharing in immutable types 519

12.3 **Using lists to represent large integers 520**
The **bigint.h** interface 521, Representing the type **bigIntADT** 521, Implementing the bigint package 524, Using the **bigint** package 529
Summary 531
Review questions 532
Programming exercises 533

13 Trees **537**

13.1 **Family trees 538**
Terminology used to describe trees 539, The recursive nature of a tree 540, Representing family trees in C 540

13.2 **Binary search trees 542**
The underlying motivation for using binary search trees 543, Finding nodes in a binary search tree 544, Inserting new nodes in a binary search tree 545, Tree traversals 549

13.3 **Balanced trees 551**
Tree-balancing strategies 552, Illustrating the AVL idea 553, Single rotations 555, Double rotations 557, Implementing the AVL algorithm 558

13.4 **Defining a general interface for binary search trees 561**
Allowing the client to define the node structure 563, Generalizing the types used for keys 570, Deleting nodes 570, Implementing the binary search tree package 572, Implementing the **symtab.h** interface using binary trees 572
Summary 580
Review questions 581
Programming exercises 583

14 Expression Trees **593**

14.1 **Overview of the interpreter 594**

14.2 **The abstract structure of expressions 597**
A recursive definition of expressions 597, Ambiguity 599, Expression trees 600, Defining an abstract interface for expressions 601

14.3 **Defining the concrete expression type 606**
Union types 606, Using tagged unions to represent expressions 608, Visualizing the concrete representation 610, Implementing the constructor and selector functions 612

14.4 **Parsing an expression 615**
Parsing and grammars 615, Parsing without precedence 617, Adding precedence to the parser 618

14.5 **Evaluating an expression 621**

Summary 623
Review questions 626
Programming exercises 627

15 Sets 641

15.1 **Sets as a mathematical abstraction 642**
Membership 643, Set operations 643, Identities on sets 645

15.2 **Designing a set interface 646**
Defining the element type 647, Writing the `set.h` interface 649, Character sets 649, Using pointer sets to avoid duplication 654

15.3 **Implementing the set package 654**

15.4 **Designing a polymorphic iterator 662**
Generalizing the prototypes of the iterator functions 663, Adding polymorphism to the iterator implementation 663, Exporting a collection type 665, Coding the iterator package 669, The `foreach` idiom 673

15.5 **Enhancing the efficiency of integer sets 674**
Characteristic vectors 674, Packed arrays of bits 675, Bitwise operators 676, Implementing characteristic vectors using the bitwise operators 678, Implementing the high-level set operations 680, Using a hybrid implementation 681

Summary 681
Review questions 683
Programming exercises 686

16 Graphs 693

16.1 **The structure of a graph 694**
Directed and undirected graphs 695, Paths and cycles 697, Connectivity 697

16.2 **Implementation strategies for graphs 698**
Representing connections using an adjacency list 700, Representing connections using an adjacency matrix 700

16.3 **Extending the graph abstraction 703**
Associating data with nodes and graphs 706, Making arcs explicit 706, Iteration and graphs 708, Layered abstractions 708, A set-based interface for graphs 709

16.4 **Graph traversals 718**
Depth-first search 719, Breadth-first search 721

16.5 **Finding minimum paths 724**
An efficient implementation of priority queues 728

Summary 732

Review questions 733

Programming exercises 735

17 Looking Ahead to Java 745

17.1 **The object-oriented paradigm 746**
The history of object-oriented programming 747, Objects, classes, and methods 748, Class hierarchies and inheritance 749

17.2 **An introduction to Java 751**
The structure of the Web 752, Applets 753, Executing a Java applet 757

17.3 **The structure of Java 758**
The syntax of Java 760, Atomic types in Java 761, Defining a new class 762, Constructor methods 763, The keyword **`this`** 764, Defining methods 765, Defining subclasses 767

17.4 **Predefined classes in Java 774**
The **`String`** class 775, The **`Hashtable`** class 776, Object wrappers for the atomic types 779, The **`Vector`** class 779, The **`Stack`** class 781

17.5 **Tools for creating interactive applets 782**
Components and containers 783, The **`action`** method 784, A sample applet for drawing shapes 785, Moving ahead 793

Summary 794

Review questions 794

Programming exercises 796

Index 809