Preface

About the Book

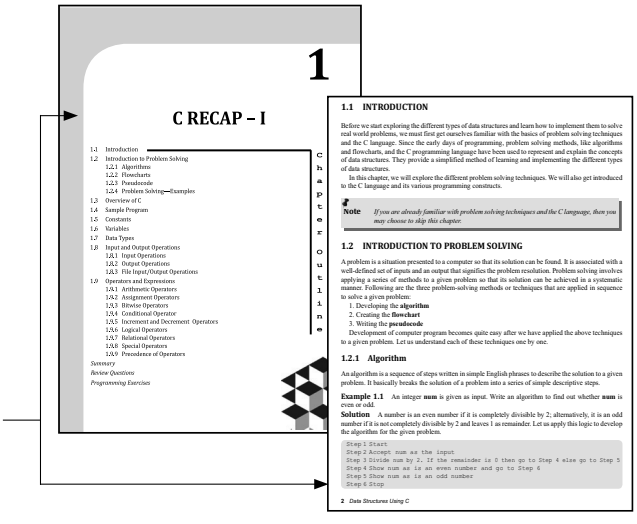
Data Structure is the way of storing data in a computer system. It allows an application to fetch and store data in the computer's memory in an efficient manner. It is very important to choose the correct type of data structure while developing a software application. **C** is one of the first programming languages that students of computer science get familiar with. It is also the language of choice while facilitating the learning of programming concepts such as data structures.

The strength of *Data Structures Using C* lies in its simple and lucid presentation of the subject which will help beginners in better understanding of the concepts. It adopts a student-friendly approach to the subject matter with many solved and unsolved examples, illustrations and well-structured C programs.

This book will prove to be a stepping stone in understanding the data structure concepts in an efficient and organized manner, and also for revisiting the fundamentals of data structure.

Salient Features of the Book

- In-depth coverage of all important topics like *Arrays, Linked lists, Stacks, Queues, Trees, Graphs, Sorting, and Searching*
- Dedicated chapter on Real Life Applications of Data Structures
- Explains run-time complexity of all algorithms
- Multiple-Choice Questions for university exams and interviews
- Innovative chapter features includes *over 400 pedagogical aids like illustrations, programs, important commands in programs, output and program analysis, note, checkpoint, key terms, solved problems, and review questions.*


1

C RECAP – I

Chapter Outline

1.1 Introduction
1.2 Introduction to Problem Solving
1.2.1 Algorithms
1.2.2 Flowcharts
1.2.3 Pseudocode
1.2.4 Problem Solving—Examples
1.3 Overview of C
1.4 Sample Program
1.5 Constants
1.6 Variables
1.7 Data Types
1.8 Input and Output Operations
1.8.1 Input Operations
1.8.2 Output Operations
1.8.3 File Input/Output Operations
1.9 Operators and Expressions
1.9.1 Arithmetic Operators
1.9.2 Assignment Operators
1.9.3 Bitwise Operators
1.9.4 Conditional Operator
1.9.5 Increment and Decrement Operators
1.9.6 Logical Operators
1.9.7 Relational Operators
1.9.8 Special Operators
1.9.9 Precedence of Operators
Summary
Review Questions
Programming Exercises

1.1 INTRODUCTION

Before we start exploring the different types of data structures and learn how to implement them to solve real world problems, we must first get ourselves familiar with the basics of problem solving techniques and the C language. Since the early days of programming, problem solving methods, like algorithms and flowcharts, and the C programming language have been used to represent and explain the concepts of data structures. They provide a simplified method of learning and implementing the different types of data structures.

In this chapter, we will explore the different problem solving techniques. We will also get introduced to the C language and its various programming constructs.

Note *If you are already familiar with problem solving techniques and the C language, then you may choose to skip this chapter.*

1.2 INTRODUCTION TO PROBLEM SOLVING

A problem is a situation presented to a computer so that its solution can be found. It is associated with a well-defined set of inputs and an output that signifies the problem resolution. Problem solving involves applying a series of methods to a given problem so that its solution can be achieved in a systematic manner. Following are the three problem-solving methods or techniques that are applied in sequence to solve a given problem:

1. Developing the **algorithm**
2. Creating the **flowchart**
3. Writing the **pseudocode**

Development of computer program becomes quite easy after we have applied the above techniques to a given problem. Let us understand each of these techniques one by one.

1.2.1 Algorithm

An algorithm is a sequence of steps written in simple English phrases to describe the solution to a given problem. It basically breaks the solution of a problem into a series of simple descriptive steps.

Example 1.1 An integer **num** is given as input. Write an algorithm to find out whether **num** is even or odd.

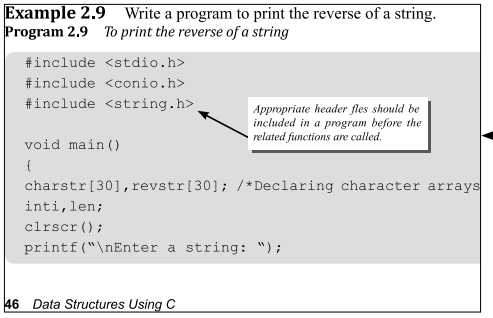
Solution A number is an even number if it is completely divisible by 2; alternatively, it is an odd number if it is not completely divisible by 2 and leaves 1 as remainder. Let us apply this logic to develop the algorithm for the given problem.

```
Step 1 Start
Step 2 Accept num as the input
Step 3 Divide num by 2. If the remainder is 0 then go to Step 4 else go to Step 5
Step 4 Show num as is an even number and go to Step 6
Step 5 Show num as is an odd number
Step 6 Stop
```

2 *Data Structures Using C*

Chapter Opening Features

At the opening of each chapter, the outline lists the major headings, followed by an introduction to the chapter. This will help students organize their study priorities.



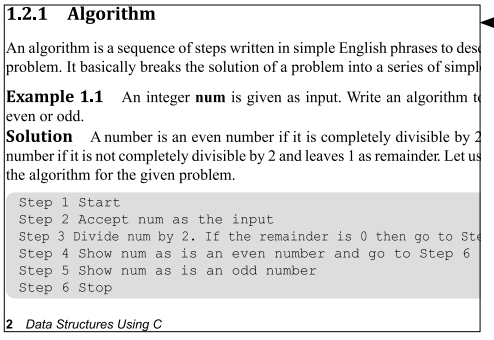
Example 2.9 Write a program to print the reverse of a string.
Program 2.9 *To print the reverse of a string*

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
charstr[30],revstr[30]; /*Declaring character arrays
inti,len;
clrscr();
printf("\nEnter a string: ");
```

Appropriate header fles should be included in a program before the related functions are called.

46 *Data Structures Using C*



1.2.1 Algorithm

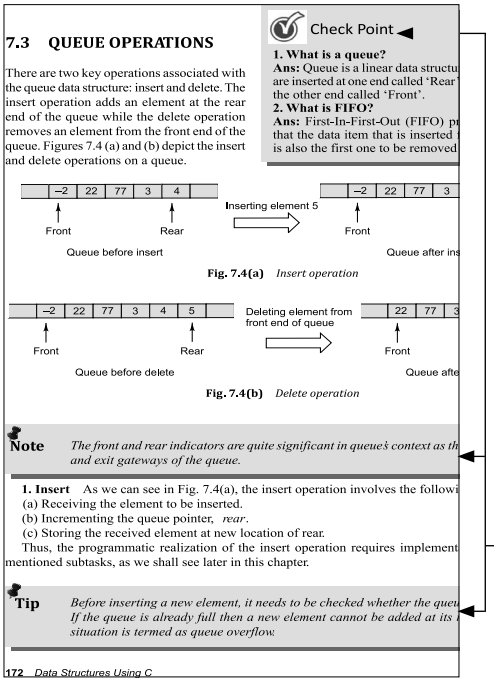
An algorithm is a sequence of steps written in simple English phrases to des
problem. It basically breaks the solution of a problem into a series of simpl

Example 1.1 An integer **num** is given as input. Write an algorithm t
even or odd.

Solution A number is an even number if it is completely divisible by 2
number if it is not completely divisible by 2 and leaves 1 as remainder. Let us
the algorithm for the given problem.

```
Step 1 Start
Step 2 Accept num as the input
Step 3 Divide num by 2. If the remainder is 0 then go to Ste
Step 4 Show num as is an even number and go to Step 6
Step 5 Show num as is an odd number
Step 6 Stop
```

2 *Data Structures Using C*



7.3 QUEUE OPERATIONS

There are two key operations associated with the queue data structure: insert and delete. The insert operation adds an element at the rear end of the queue while the delete operation removes an element from the front end of the queue. Figures 7.4 (a) and (b) depict the insert and delete operations on a queue.

Check Point

1. What is a queue?
Ans: Queue is a linear data structu
are inserted at one end called 'Rear'
the other end called 'Front'.
2. What is FIFO?
Ans: First-In-First-Out (FIFO) pr
that the data item that is inserted
is also the first one to be removed.

Fig. 7.4(a) *Insert operation*

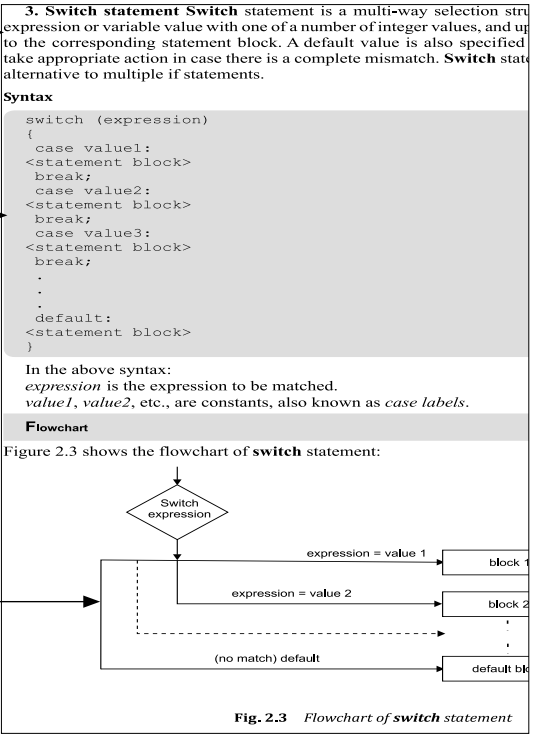
Fig. 7.4(b) *Delete operation*

Note *The front and rear indicators are quite significant in queue's context as th
and exit gateways of the queue.*

1. Insert As we can see in Fig. 7.4(a), the insert operation involves the followi
(a) Receiving the element to be inserted.
(b) Incrementing the queue pointer, *rear*.
(c) Storing the received element at new location of rear.
Thus, the programmatic realization of the insert operation requires implement
mentioned subtasks, as we shall see later in this chapter.

Tip *Before inserting a new element, it needs to be checked whether the que
If the queue is already full then a new element cannot be added at its
situation is termed as queue overflow.*

172 *Data Structures Using C*



3. Switch statement Switch statement is a multi-way selection stru
expression or variable value with one of a number of integer values, and up
to the corresponding statement block. A default value is also specified
take appropriate action in case there is a complete mismatch. **Switch** stat
alternative to multiple if statements.

Syntax

```
switch (expression)
{
 case value1:
<statement block>
 break;
 case value2:
<statement block>
 break;
 case value3:
<statement block>
 break;
 .
 .
 .
 default:
<statement block>
}
```

In the above syntax:
expression is the expression to be matched.
value1, *value2*, etc., are constants, also known as *case labels*.

Flowchart

Figure 2.3 shows the flowchart of **switch** statement:

Fig. 2.3 *Flowchart of* ***switch*** *statement*

In-chapter Features

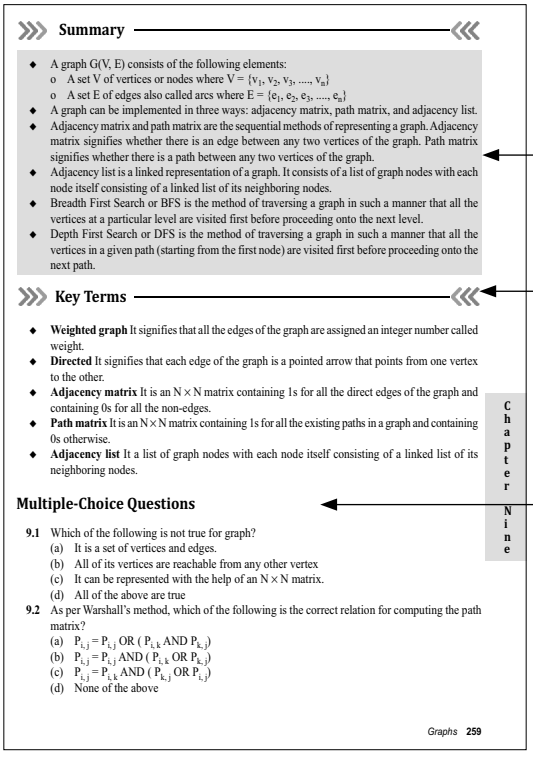
Features like algorithms, pseudocodes, flowcharts and programs emphasize on a point or help teach a concept. Commands in bold draw students' attention to a particular section in the program.

Other Significant Features

Notes, Tips and Checkpoints are designed to provide extra information or alternative views or results or interesting snippets of information related to the content of the chapter.

Chapter-end Features

Summary reviews the concepts while a list of key terms helps identify the vocabulary students need to understand the concepts presented in the chapter. Students can assess their knowledge by answering the basic review questions, programming exercises and multiple-choice questions.



Summary

- A graph G(V, E) consists of the following elements:
  - o A set V of vertices or nodes where V = {v_1, v_2, v_3, ..., v_n}
  - o A set E of edges also called arcs where E = {e_1, e_2, e_3, ..., e_n}
- A graph can be implemented in three ways: adjacency matrix, path matrix, and adjacency list.
- Adjacency matrix and path matrix are the sequential methods of representing a graph. Adjacency matrix signifies whether there is an edge between any two vertices of the graph. Path matrix signifies whether there is a path between any two vertices of the graph.
- Adjacency list is a linked representation of a graph. It consists of a list of graph nodes with each node itself consisting of a linked list of its neighboring nodes.
- Breadth First Search or BFS is the method of traversing a graph in such a manner that all the vertices at a particular level are visited first before proceeding onto the next level.
- Depth First Search or DFS is the method of traversing a graph in such a manner that all the vertices in a given path (starting from the first node) are visited first before proceeding onto the next path.

Key Terms

- **Weighted graph** It signifies that all the edges of the graph are assigned an integer number called weight.
- **Directed** It signifies that each edge of the graph is a pointed arrow that points from one vertex to the other.
- **Adjacency matrix** It is an N × N matrix containing 1s for all the direct edges of the graph and containing 0s for all the non-edges.
- **Path matrix** It is an N × N matrix containing 1s for all the existing paths in a graph and containing 0s otherwise.
- **Adjacency list** It a list of graph nodes with each node itself consisting of a linked list of its neighboring nodes.

Multiple-Choice Questions

9.1 Which of the following is not true for graph?
(a) It is a set of vertices and edges.
(b) All of its vertices are reachable from any other vertex
(c) It can be represented with the help of an N × N matrix.
(d) All of the above are true

9.2 As per Warshall's method, which of the following is the correct relation for computing the path matrix?
(a) $P_{i,j} = P_{i,j}$ OR ($P_{i,k}$ AND $P_{k,j}$)
(b) $P_{i,j} = P_{i,j}$ AND ($P_{i,k}$ OR $P_{k,j}$)
(c) $P_{i,j} = P_{i,k}$ AND ($P_{k,j}$ OR $P_{i,j}$)
(d) None of the above

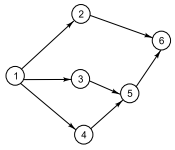
Graphs 259

Chapter Nine

9.3 As per modified Warshall's algorithm, which of the following is the correct relation for computing the shortest path between two vertices in a graph?
(a) $SP_{i,j}$ = Minimum of ($SP_{i,j}$, $SP_{i,k} + SP_{k,j}$)
(b) $SP_{i,j}$ = Maximum of ($SP_{i,j}$, $SP_{i,k} + SP_{k,j}$)
(c) $SP_{i,j}$ = Minimum of ($SP_{i,k}$, $SP_{k,j} + SP_{i,j}$)
(d) None of the above

9.4 The number of edges incident on a vertex is referred as _____.
(a) Degree
(b) Indegree
(c) Order
(d) Outdegree

9.5 Identify the BFS path for the following graph:
(a) 1–2–3–4–6–5
(b) 1–4–3–2–6–5
(c) 1–2–3–4–5–6
(d) None of the above



Review Questions

9.1 What is a graph? Explain with an example.
9.2 List and explain any five key terms associated with graphs.
9.3 What are the different methods of representing a graph?
9.4 What is an adjacency matrix? How can you derive a path matrix from an adjacency matrix?
9.5 Explain adjacency list implementation of a graph with the help of an example.
9.6 What is the significance of computing the shortest path in a graph? Explain with the help of an example.
9.7 Write the modified Warshall's algorithm for computing the shortest path between two nodes of a graph.
9.8 What is BFS? Explain with the help of an example.
9.9 What is DFS? Explain with the help of an example.

Programming Exercises

9.1 Write a C function to deduce the adjacency matrix for a given directed graph G.
9.2 Write a C function that takes as input the adjacency matrix and applies Warshall's algorithm to generate the corresponding path matrix.
9.3 Write a C program to implement a 3-node directed graph using adjacency list.
9.4 Write a C function that takes as input the path matrix and applies the shortest path algorithm to generate the corresponding shortest path matrix.

Answers to Multiple-Choice Questions

9.1 (b) 9.2 (a) 9.3 (a) 9.4 (b) 9.5 (c)

260 Data Structures Using C

Chapter Organization

This book is organized into 11 chapters, which explain concepts like Arrays, Stacks, Queues, Linked Lists, Trees and Graphs.

Chapters 1 and 2 provide a quick recap to the C programming language. **Chapter 3** introduces algorithm and its related concepts. It also provides a brief introduction to the different types of data structures. **Chapter 4** discusses one of the commonly used derived data types, i.e., array and explains how it is used as a data structure in different programming situations. **Chapter 5** explains the concept of linked list along with its different variants. **Chapters 6 and 7** elucidates the restricted data structures, stacks and queues. These data structures are of great importance in programming situations because of the specific restrictions that they apply on insertion and deletion of data elements. **Chapters 8 and 9**

explain the non-linear data structures trees and graphs and their related operations. These chapters also explain the various algorithms that are used to traverse these data structures. **Chapter 10** introduces two of the most common computing operations, i.e., searching and sorting. It explains various searching and sorting techniques along with their related advantages and disadvantages. Finally, **Chapter 11** explains how all the data structures taught previously are put into use for solving mathematical and other real-world problems.

Web Supplements

Following materials can be accessed at ***http://www.mhhe.com/balagurusamy/dsuc***

- *PowerPoint slides*
- *Computer programs for labs*
- *Links for additional resources*

Acknowledgements

I would like to thank the following reviewers for their suggestions in improving the script:

Shashank Dwivedi	*UCER, Allahabad, Uttar Pradesh*
Rajiv Pandey	*Amity University Lucknow Campus, Lucknow, Uttar Pradesh*
Mahua Banerjee	*Xavier Institute of Social Service, Ranchi, Jharkhand*
Sameer Bhave	*Indore Professional Studies Academy, Indore, Madhya Pradesh*
D Lakshmi, Adithya	*Institute of Technology, Coimbatore, Tamil Nadu*
A Sharada	*G Narayanamma Institute of Technology and Science, Hyderabad, Andhra Pradesh*

Sincere thanks to the editorial team of McGraw Hill Education (India) for their support and cooperation.

Publisher's Note

Suggestions and constructive criticism always go a long way in enhancing any endeavor. We request all readers to email us their valuable comments/views/feedback for the betterment of the book at *tmh.csefeedback@gmail.com* mentioning the title and the author's name in the subject line.

Please report any piracy spotted by you as well!