



A T M E
College of Engineering

13th KM Stone, Bannur Road, Mysore - 560 028

**Department of CSE–Artificial Intelligence & Machine Learning
(Academic Year 2026-27)**



LABORATORY MANUAL

**SUBJECT: DATA STRUCTURES LABORATORY
SUB CODE: 1BCSL306**

SEMESTER: III

SCHEME: 2025

Prepared By

**Ms. GEETHA.B
Instructor**

Verified By

**Ms. Likitha D
Assistant. Professor
& Ms. Harshitha V
Assistant. Professor
Dept. of CSE-AI &ML**

Approved by

**Dr. ANIL KUMAR C.J
Assoc. Professor & Head
Dept. of CSE-AI &ML**

Institute Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Institute Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

Department Vision

- To impart technical education in the field of Artificial intelligence and machine learning of topnotch quality with a high level of professional competence, social obligation, and global cognizance among the students.

Department Mission

- To impart technical education that is up to date, relevant and makes students to compete at global level
- Fostering an ambiance where students can adopt the suitable moral, intellectual, emotional, and physical attributes to shine as the leaders of tomorrow's society.
- To strive to meet ever higher educational standard.

Program Outcomes (PO's)

PO1.Engineering Knowledge: Apply fundamental knowledge to solve complex engineering problems.

PO1.Engineering Knowledge: Apply fundamental knowledge to solve complex engineering problems

PO3.Design/Development of Solutions: Create solutions that meet specific needs, considering health, safety, and environmental factors.

PO4.Conduct Investigations of Complex Problems: Use research-based methods to design experiments, analyze, and interpret data.

PO5.Engineering Tool Usage: Select and apply modern techniques and IT tools to engineering activities.

PO6.The Engineer and the World: Understand the impact of professional engineering solutions on society and the environment.

PO7.Ethics: Commit to professional ethics and responsibilities.

PO8.Individual and Collaborative Team Work: Function effectively as an individual or team member in multidisciplinary settings.

PO9.Communication: Communicate effectively regarding complex engineering activities with the engineering community and society.

PO10.Project Management and Finance: Apply management and financial principles to project work.

PO11. Life-Long Learning: Engage in independent and life-long learning in the context of technological change.

Program Educational Objectives (PEO's):

PEO1: Graduates will be able to hone their problem-solving abilities and capacity to offer solutions to challenges that arise in the actual world.

PEO2: Able to design and develop AI based solutions to real-world problems in a business, research, or social environment.

PEO3: Graduates shall acquire and inculcate corporate culture, core attributes, and leadership qualities as well as professional etiquettes and lifelong learning.

Program Specific Outcomes (PSO's)

PSO1: Ability to design and develop artificial intelligent based solutions by applying optimal algorithms to solve real world issues.

PSO2: Ability to apply suitable AI tools and techniques to offer solutions in the various domains of engineering

DATA STRUCTURES LABORATORY		Scheme	2025
Course Code	1BCSL306	Semester	3
Teaching Hours/Week (L:T:P)	0:0:2	CIE Marks	50
Total Hours of Pedagogy L:T:P:SL:TW	0:0:28:2:30	SEE Marks	50
Credits	1	Total Marks	100
Examination type (SEE)	Practical	Exam Hours	03

PRACTICAL COMPONENT

PART – A FIXED SET OF EXPERIMENTS

1. Develop a Program in C for the following:
 - a. Define a structure Book with the following fields: Book_ID, Title, Author, Price, Availability Status (Available/Issued)
 - b. Dynamically allocate memory to store the details of N books, where N is entered by the user.
 - c. Implement the following functions:
 - i. create() – To allocate memory and read the details of N books and store them in the allocated memory.
 - ii. display() – To display the details of all books available in the library.
 - iii. search() – To search for a book using Book ID and display the book details if found.
 - iv. issueBook() – To issue a book by updating its availability status and display an appropriate message.
 - v. returnBook() – To return an issued book by updating its availability status and display an appropriate message.
 - d. Provide a menu-driven interface with the following options:
 - i. Add Book Records
 - ii. Display All Book Records
 - iii. Search Book by Book ID
 - iv. Issue a Book
 - v. Return a Book
 - vi. Exit
2. Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)
 - a. Push an Element on to Stack
 - b. Pop an Element from Stack
 - c. Demonstrate how Stack can be used to check Palindrome
 - d. Demonstrate Overflow and Underflow situations on Stack
 - e. Display the status of Stack
 - f. Exit

Support the program with appropriate functions for each of the above operations
3. Develop a program in C for simulating a Printer Queue using Queue data structure with the following operations:
 - a. Add a print job to the Printer Queue.
 - b. Process (delete) a print job from the Printer Queue
 - c. Display all pending print jobs in the Queue.
 - d. Display the number of print jobs currently waiting in the Queue.
 - e. Demonstrate Queue overflow and underflow conditions.
 - f. Exit
4. Develop a menu-driven Program in C for the following operations on a Singly Linked List of Student Records:
 - a. Create a linked list using front insertion.
 - b. Display all student records and count the number of nodes.

- c. Insert and delete nodes at the beginning and end of the list.
 - d. Search for a student record using USN.
5. Develop a Program in C for the following:
- a. Create a Binary Tree by inserting integer elements entered by the user in level order.
 - b. Implement the following functions:
 - i. **createTree ()** – Create a binary tree using dynamic memory allocation.
 - ii. **preorder ()** – Display the elements using Preorder Traversal.
 - iii. **inorder ()** – Display the elements using Inorder Traversal.
 - iv. **postorder ()** – Display the elements using Postorder Traversal.
 - c. Provide a menu-driven interface with the following options:
 - i. Create Binary Tree
 - ii. Display Preorder Traversal
 - iii. Display Inorder Traversal
 - iv. Display Postorder Traversal
 - v. Exit
6. Develop a Program in C for the following operations on Graph(G) of Cities
- a. Create an Adjacency Matrix representing the graph of N cities
 - b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method.

PART – B OPEN ENDED EXPERIMENTS

Open-ended experiments are a type of laboratory activity where the outcome is not predetermined and students are given the freedom to explore, design, and conduct the experiment based on the problem statements as per the concepts defined by the course coordinator. It encourages creativity, critical thinking, and inquiry-based learning.

1. Develop a C program to:
 - a. Read two sparse matrices of the same order.
 - b. Convert both into 3-tuple representation.
 - c. Perform matrix addition.
 - d. Display the resultant matrix in both normal matrix form and 3-tuple form.
2. Develop a C-based Expression Conversion tool that uses stack data structure to convert arithmetic expressions from infix notation to postfix notation. The tool should support alphanumeric operands, parentheses, and the operators (+, -, *, /, %, ^). Analyze the correctness of the conversion by testing various valid expressions.
3. Develop a menu-driven Program in C for the following operations on Circular Queue (integer/floating point data):
 - a. Insert an element into the Circular Queue.
 - b. Delete an element from the Circular Queue.
 - c. Display the contents of the Circular Queue.
 - d. Demonstrate overflow and underflow conditions.
 - e. Exit.
4. Develop a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo etc.
 - a. Create a DLL of N Employees Data by using end insertion.
 - b. Display the status of DLL and count the number of nodes in it
 - c. Perform Insertion and Deletion at End of DLL
 - d. Perform Insertion and Deletion at Front of DLL
 - e. Demonstrate how this DLL can be used as Double Ended Queue
 - f. Exit
5. Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers.
 - a. Create a BST of N Integers
 - b. Traverse the BST in Inorder, Preorder and Post Order

- c. Search the BST for a given element (KEY) and report the appropriate message
- d. Exit

6. Develop a Program in C for an Employee Record Management System using Hashing with the following operations:

- a. Create a hash table to store employee records using Employee ID as the key.
- b. Insert employee records into the hash table using the Hash function $H: K \rightarrow L$ as $H(K) = K \bmod m$
- c. Resolve collisions using Linear Probing.
- d. Search for an employee record using a given Employee ID and display the details.

Display the contents of the hash table along with the corresponding memory locations.

Course outcome (Course Skill Set)

At the end of the course, the student will be able to:

1. Develop C programs to implement fundamental data structures and their operations for solving computational problems.
2. Make use of appropriate data structures to organize, store, and manipulate data for solving real-world problems using C.
3. Build a laboratory record consisting of problem statements, algorithm/design, implementation, program outputs, and result with appropriate inferences.

Assessment Structure:

The assessment for each course is equally divided between Continuous Internal Evaluation (CIE) and the Semester End Examination (SEE), with each component carrying **50% weightage** (i.e., 50 marks each).

The CIE marks awarded shall be based on the continuous evaluation of the laboratory report using a defined set of rubrics. Each experiment report can be evaluated for 30 marks. Total marks for the all report shall be scaled down to 30 marks. The laboratory test (duration 03 hours) at the end of the last week of the semester /after completion of all the experiments (whichever is early) shall be conducted for 50 marks and Marks scored shall be scaled down to 20 marks. Final average of marks for report and test shall be out of 50 marks.

For both CIE and SEE, the student is required to conduct one experiment each from both Part A and Part B.

- To qualify and become eligible to appear for SEE, in the CIE component, a student must secure a minimum of 40% of 50 marks, i.e., 20 marks.
- To pass the SEE component, a student must secure a minimum of 35% of 50 marks, i.e., 18 marks.

A student is deemed to have successfully completed the course if the combined total of CIE and SEE is at least 40 out of 100 marks.

CONTENTS

Sl. NO	EXPERIMENTS	Page No
	INTRODUCTION	1-2
PART – A FIXED SET OF EXPERIMENTS		
1	Program-1: Develop a Program in C for the following: a. Define a structure Book with the following fields: Book_ID, Title, Author, Price, Availability Status (Available/Issued) b. Dynamically allocate memory to store the details of N books, where N is entered by the user. c. Implement the following functions: i. create() – To allocate memory and read the details of N books and store them in the allocated memory. ii. display() – To display the details of all books available in the library. iii. search() – To search for a book using Book ID and display the book details if found. iv. issueBook() – To issue a book by updating its availability status and display an appropriate message. v. returnBook() – To return an issued book by updating its availability status and display an appropriate message. d. Provide a menu-driven interface with the following options: i. Add Book Records ii. Display All Book Records iii. Search Book by Book ID iv. Issue a Book v. Return a Book vi. Exit	3-9
2	Program-2: Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations	10-18
3	Program-3: Develop a program in C for simulating a Printer Queue using Queue data structure with the following operations: a. Add a print job to the Printer Queue. b. Process (delete) a print job from the Printer Queue c. Display all pending print jobs in the Queue. d. Display the number of print jobs currently waiting in the Queue. e. Demonstrate Queue overflow and underflow conditions. f. Exit	19-24
4	Program-4: Develop a menu-driven Program in C for the following operations on a Singly Linked List of Student Records: a. Create a linked list using front insertion. b. Display all student records and count the number of nodes. c. Insert and delete nodes at the beginning and end of the list. d. Search for a student record using USN.	25-30

5	Program-5: Develop a Program in C for the following: a. Create a Binary Tree by inserting integer elements entered by the user in level order. b. Implement the following functions: - i.createTree () – Create a binary tree using dynamic memory allocation. - ii.preorder () – Display the elements using Preorder Traversal. - iii.inorder () – Display the elements using Inorder Traversal. - iv.postorder () – Display the elements using Postorder Traversal. c. Provide a menu-driven interface with the following options: - i.Create Binary Tree - ii.Display Preorder Traversal - iii.Display Inorder Traversal - iv.Display Postorder Traversal v. Exit	31-36
6	Program-6: Develop a Program in C for the following operations on Graph(G) of Cities a. Create an Adjacency Matrix representing the graph of N cities b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method.	37-40
PART – B OPEN ENDED EXPERIMENTS		
1	Program-1: Develop a C program to: a. Read two sparse matrices of the same order. b. Convert both into 3-tuple representation. c. Perform matrix addition. d. Display the resultant matrix in both normal matrix form and 3-tuple form.	41-47
2	Program-2: Develop a C-based Expression Conversion tool that uses stack data structure to convert arithmetic expressions from infix notation to postfix notation. The tool should support alphanumeric operands, parentheses, and the operators (+, -, *, /, %, ^). Analyze the correctness of the conversion by testing various valid expressions.	48-55
3	Program-3: Develop a menu-driven Program in C for the following operations on Circular Queue (integer/floating point data): a. Insert an element into the Circular Queue. b. Delete an element from the Circular Queue. c. Display the contents of the Circular Queue. d. Demonstrate overflow and underflow conditions. e. Exit.	56-63
4	Program-4: Develop a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo etc. a. Create a DLL of N Employees Data by using end insertion. b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue f. Exit	64-77
5	Program-5: Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers. a. Create a BST of N Integers b. Traverse the BST in Inorder, Preorder and Post Order c. Search the BST for a given element (KEY) and report the appropriate message d. Exit	78-83

6	Program-6: Develop a Program in C for an Employee Record Management System using Hashing with the following operations: a. Create a hash table to store employee records using Employee ID as the key. b. Insert employee records into the hash table using the Hash function $H: K \rightarrow L$ as $H(K)=K \bmod m$ c. Resolve collisions using Linear Probing. d. Search for an employee record using a given Employee ID and display the details. Display the contents of the hash table along with the corresponding memory locations.	84-89
	VIVA QUESTIONS WITH ANSWERS	90-94

INTRODUCTION

Data Structures is about how data can be stored in different structures.

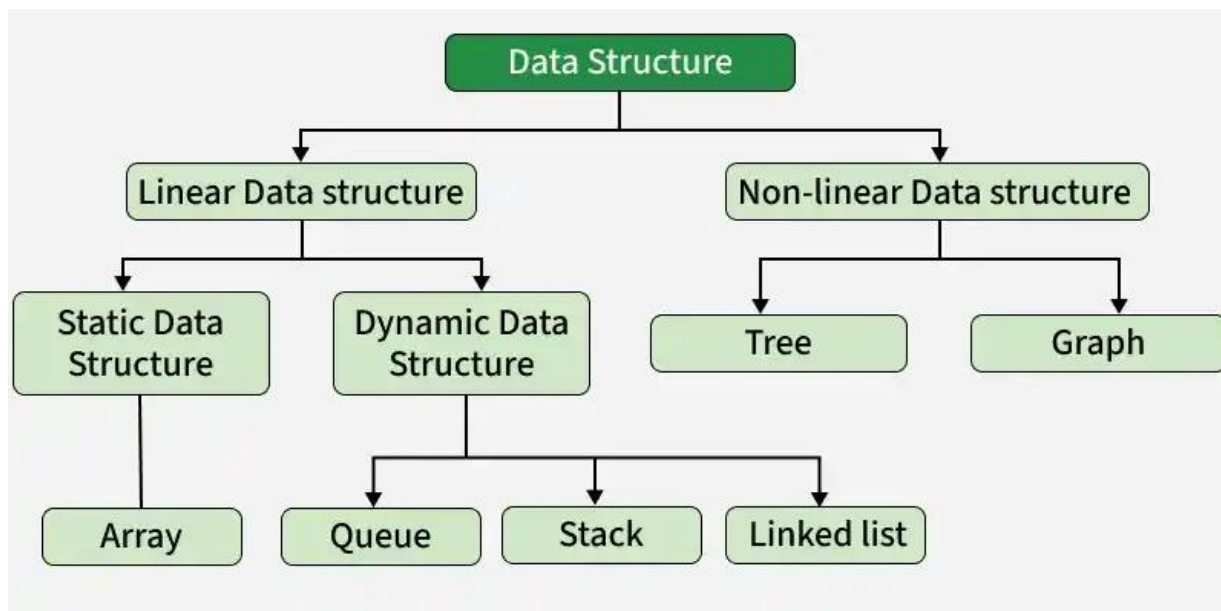
Algorithms is about how to solve different problems, often by searching through and manipulating data structures.

Theory about Data Structures and Algorithms (DSA) helps us to use large amounts of data to solve problems efficiently.

What is Data Structure?

A data structure is a way of organizing and storing data in a computer so that it can be accessed and used efficiently. It refers to the logical or mathematical representation of data, as well as the implementation in a computer program.

Data Structure



A **static data structure** is an organization or collection of data in memory that has a **fixed size** determined at **compile-time**

A **dynamic data structure** is a collection of data in memory that has the flexibility to **grow or shrink in size during program execution**

The academic curriculum for DSA is broadly divided into three main pillars:

1. Performance Measurement

Before writing code, you learn how to evaluate it. You will study **Time Complexity** (how fast your code runs) and **Space Complexity** (how much memory it uses) using **Big O Notation**.

2. Types of Data Structures

You will implement and manipulate two primary types of data structures:

- **Linear Data Structures:** Elements are arranged sequentially.
 - Arrays: Fixed-size sequential elements.
 - Linked Lists: Nodes linked dynamically via pointers.
 - Stacks: Last-In-First-Out (LIFO) structures used in undo operations.

- Queues: First-In-First-Out (FIFO) structures used in task scheduling.
- **Non-Linear Data Structures:** Elements are arranged hierarchically.
 - Trees: Hierarchical parent-child nodes (like File Systems).
 - Graphs: Interconnected networks of nodes (like Facebook friend networks).

3. Standard Algorithms

You will study standard problem-solving strategies:

- **Searching:** Finding specific data (e.g., Binary Search).
- **Sorting:** Arranging data orderly (e.g., Merge Sort, Quick Sort).
- **Advanced Paradigms:** Greedy Algorithms, Divide and Conquer, and Dynamic Programming.

A data structure helps us to understand the relationship of one element to another element and organize it within the memory. It is a mathematical or logical representation or organization of data in memory. Data structures are extensively applied in the following areas:

1. Compiler Design
2. Database Management Systems (DBMS)
3. Artificial Intelligence
4. Network and Numerical Analysis
5. Statistical Analysis Packages
6. Graphics
7. Operating Systems (OS)
8. Simulations

Applications of Data Structures

Data structures are used in a wide range of computer programs and applications, including:

- **Databases:** Data structures are used to organize and store data in a database, allowing for efficient retrieval and manipulation.
- **Operating systems:** Data structures are used in the design and implementation of operating systems to manage system resources, such as memory and files.
- **Computer graphics:** Data structures are used to represent geometric shapes and other graphical elements in computer graphics applications.
- **Artificial intelligence:** Data structures are used to represent knowledge and information in artificial intelligence systems.

Advantages of Data Structures:

The use of data structures provides several advantages, including:

- **Efficiency:** Data structures allow for efficient storage and retrieval of data, which is important in applications where performance is critical.
- **Flexibility:** Data structures provide a flexible way to organize and store data, allowing for easy modification and manipulation.
- **Reusability:** Data structures can be used in multiple programs and applications, reducing the need for redundant code.
- **Maintainability:** Well-designed data structures can make programs easier to understand, modify, and maintain over time.

1. Develop a Program in C for the following:

- a. Define a structure Book with the following fields: Book_ID, Title, Author, Price, a availability Status (Available/Issued)
- b. Dynamically allocate memory to store the details of N books, where N is entered by the user.

a. Implement the following functions:

- i. create () – To allocate memory and read the details of N books and store them in the allocated memory.
- ii. display () – To display the details of all books available in the library.
- iii. search () – To search for a book using Book ID and display the book details if found.
- iv. issue Book () – To issue a book by updating its availability status and display an appropriate message.
- v. return Book () – To return an issued book by updating its availability status and display an appropriate message.

b. Provide a menu-driven interface with the following options:

- i. Add Book Records
- ii. Display All Book Records
- iii. Search Book by Book ID
- iv. Issue a Book
- v. Return a Book
- vi. Exit

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Define the Book structure
typedef struct {
    int book_id;
    char title[100];
    char author[50];
    float price;
    int is_available; // 1 for Available, 0 for Issued
} Book;
// Function prototypes
Book* create(int *n);
void display(Book *books, int n);
void search(Book *books, int n);
void issueBook(Book *books, int n);
void returnBook(Book *books, int n);
int main()
{
    Book *library = NULL;
    int n = 0;
    int choice;
    while (1)
    {
        printf("\n===== LIBRARY MANAGEMENT SYSTEM =====\n");
        printf("1. Add Book Records\n");
        printf("2. Display All Book Records\n");
        printf("3. Search Book by Book ID\n");
        printf("4. Issue a Book\n");
```

```
printf("5. Return a Book\n");
printf("6. Exit\n");
printf("Enter your choice (1-6): ");
scanf("%d", &choice);
switch (choice)
{
    case 1:
        if (library != NULL)
        {
            free(library); // Free previously allocated memory if overriding
        }
        library = create(&n);
        break;
    case 2:
        display(library, n);
        break;
    case 3:
        search(library, n);
        break;
    case 4:
        issueBook(library, n);
        break;
    case 5:
        returnBook(library, n);
        break;
    case 6:
        printf("Exiting program. Thank you!\n");
        if (library != NULL)
        {
            free(library); // Clean up memory before exiting
        }
        return 0;
    default:
        printf("Invalid choice! Please enter a number between 1 and 6.\n");
}
}
return 0;
}
// Function to allocate memory and read details of N books
Book* create(int *n)
{
    int i;
    printf("Enter the number of books: ");
    scanf("%d", n);
    if (*n <= 0)
    {
        printf("Invalid number of books!\n");
        return NULL;
    }
}
```

```
}
// Dynamic memory allocation
Book *books = (Book*)malloc((*n) * sizeof(Book));
if (books == NULL)
{
    printf("Memory allocation failed!\n");
    exit(1);
}
for (i = 0; i < *n; i++)
{
    printf("\nEnter details for Book %d:\n", i + 1);
    printf("Book ID: ");
    scanf("%d", &books[i].book_id);
    getchar(); // Clear newline character from buffer
    printf("Title: ");
    fgets(books[i].title, sizeof(books[i].title), stdin);
    books[i].title[strcspn(books[i].title, "\n")] = 0;
    printf("Author: ");
    fgets(books[i].author, sizeof(books[i].author), stdin);
    books[i].author[strcspn(books[i].author, "\n")] = 0;
    printf("Price: ");
    scanf("%f", &books[i].price);
    books[i].is_available = 1; // Default status is set to 'Available'
}
printf("\n%d book records created successfully!\n", *n);
return books;
}
// Function to display all book details
void display(Book *books, int n)
{
    int i;
    if (books == NULL || n == 0)
    {
        printf("\nNo records found! Please add books first.\n");
        return;
    }
    printf("\n-----\n");
    printf("%-10s %-25s %-20s %-10s %-15s\n",
        "Book ID", "Title", "Author", "Price", "Status");
    printf("-----\n");
    for (i = 0; i < n; i++)
    {
        printf("%-10d %-25s %-20s %-10.2f %-15s\n", books[i].book_id, books[i].title, books[i].author,
            books[i].price, books[i].is_available ? "Available" : "Issued");
    }
    printf("-----\n");
}
// Function to search for a book using Book ID
```

```
void search(Book *books, int n)
{
    int i;
    if (books == NULL || n == 0)
    {
        printf("\nNo records found! Please add books first.\n");
        return;
    }
    int id, found = 0;
    printf("Enter Book ID to search: ");
    scanf("%d", &id);
    for (i = 0; i < n; i++)
    {
        if (books[i].book_id == id)
        {
            printf("\nBook Found!\n");
            printf("ID: %d\nTitle: %s\nAuthor: %s\nPrice: %.2f\nStatus: %s\n", books[i].book_id,
                books[i].title, books[i].author, books[i].price, books[i].is_available ? "Available" : "Issued");
            found = 1;
            break;
        }
    }
    if (!found)
    {
        printf("\nBook with ID %d not found.\n", id);
    }
}

// Function to issue a book
void issueBook(Book *books, int n)
{
    int i;
    if (books == NULL || n == 0)
    {
        printf("\nNo records found! Please add books first.\n");
        return;
    }
    int id, found = 0;
    printf("Enter Book ID to issue: ");
    scanf("%d", &id);
    for (i = 0; i < n; i++)
    {
        if (books[i].book_id == id)
        {
            found = 1;
            if (books[i].is_available == 1)
            {
                books[i].is_available = 0; // Update status to Issued
                printf("\nSuccess: Book '%s' has been issued successfully!\n", books[i].title);
            }
        }
    }
}
```

```
        }
        else
        {
            printf("\nFailure: Book '%s' is already issued to someone else.\n",books[i].title);
        }
        break;
    }
}
if (!found)
{
    printf("\nBook with ID %d not found.\n", id);
}
}
// Function to return an issued book
void returnBook(Book *books, int n)
{
    int i;
    if (books == NULL || n == 0)
    {
        printf("\nNo records found! Please add books first.\n");
        return;
    }
    int id, found = 0;
    printf("Enter Book ID to return: ");
    scanf("%d", &id);
    for (i = 0; i < n; i++)
    {
        if (books[i].book_id == id)
        {
            found = 1;
            if (books[i].is_available == 0)
            {
                books[i].is_available = 1; // Update status to Available
                printf("\nSuccess: Book '%s' has been returned successfully!\n",books[i].title);
            }
            else
            {
                printf("\nFailure: Book '%s' was not issued.It is already available in the library.\n",books[i].title);
            }
            break;
        }
    }
    if (!found)
    {
        printf("\nBook with ID %d not found.\n", id);
    }
}
```

OUTPUT:

===== LIBRARY MANAGEMENT SYSTEM =====

1. Add Book Records
2. Display All Book Records
3. Search Book by Book ID
4. Issue a Book
5. Return a Book
6. Exit

Enter your choice (1-6): 1

Enter the number of books: 2

Enter details for Book 1:

Book ID: 101

Title: The C Programming Language

Author: Dennis Ritchie

Price: 450.50

Enter details for Book 2:

Book ID: 102

Title: Clean Code

Author: Robert Martin

Price: 650.00

2 book records created successfully!

===== LIBRARY MANAGEMENT SYSTEM =====

1. Add Book Records
2. Display All Book Records
3. Search Book by Book ID
4. Issue a Book
5. Return a Book
6. Exit

Enter your choice (1-6): 2

Book ID	Title	Author	Price	Status
101	The C Programming Language	Dennis Ritchie	450.50	Available
102	Clean Code	Robert Martin	650.00	Available

===== LIBRARY MANAGEMENT SYSTEM =====

1. Add Book Records
2. Display All Book Records
3. Search Book by Book ID
4. Issue a Book
5. Return a Book
6. Exit

Enter your choice (1-6): 4

Enter Book ID to issue: 101

Success: Book 'The C Programming Language' has been issued successfully!

===== LIBRARY MANAGEMENT SYSTEM =====

1. Add Book Records
2. Display All Book Records
3. Search Book by Book ID
4. Issue a Book
5. Return a Book
6. Exit

Enter your choice (1-6): 2

```
-----
Book ID  Title                Author          Price    Status
-----
101      The C Programming Language Dennis Ritchie  450.50   Issued
102      Clean Code                 Robert Martin   650.00   Available
-----
```

===== LIBRARY MANAGEMENT SYSTEM =====

1. Add Book Records
2. Display All Book Records
3. Search Book by Book ID
4. Issue a Book
5. Return a Book
6. Exit

Enter your choice (1-6): 3

Enter Book ID to search: 102

Book Found!

ID: 102

Title: Clean Code

Author: Robert Martin

Price: 650.00

Status: Available

===== LIBRARY MANAGEMENT SYSTEM =====

1. Add Book Records
2. Display All Book Records
3. Search Book by Book ID
4. Issue a Book
5. Return a Book
6. Exit

Enter your choice (1-6): 5

Enter Book ID to return: 101

Success: Book 'The C Programming Language' has been returned successfully!

===== LIBRARY MANAGEMENT SYSTEM =====

1. Add Book Records
2. Display All Book Records
3. Search Book by Book ID
4. Issue a Book
5. Return a Book
6. Exit

Enter your choice (1-6): 6

Program terminated successfully.

2. Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

- a. Push an Element on to Stack**
- b. Pop an Element from Stack**
- c. Demonstrate how Stack can be used to check Palindrome**
- d. Demonstrate Overflow and Underflow situations on Stack**
- e. Display the status of Stack**
- f. Exit**

Support the program with appropriate functions for each of the above operations

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int s[MAX];
int top = -1;
void push(int item);
int pop();
void palindrome();
void display();
void main()
{
    int choice, item;
    while (1)
    {
        printf("\n\n\n~~~~~Menu~~~~~ : ");
        printf("\n=>1.Push an Element to Stack and Overflow demo ");
        printf("\n=>2.Pop an Element from Stack and Underflow demo");
        printf("\n=>3.Palindrome demo ");
        printf("\n=>4.Display ");
        printf("\n=>5.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("\nEnter an element to be pushed: ");
                scanf("%d", &item);
                push(item);
                break;
            case 2:
                item = pop();
                if (item != -1)
                    printf("\nElement popped is: %d", item);
                break;
            case 3:
                palindrome();
                break;
            case 4:
```

```
        display();
        break;
    case 5:
        exit(1);
    default:
        printf("\nPlease enter valid choice ");
        break;
    }
}
}
void push(int item)
{
    if (top == MAX - 1)
    {
        printf("\n~~~~~Stack overflow~~~~~");
        return;
    }
    top = top + 1;
    s[top] = item;
}
int pop()
{
    int item;
    if (top == -1)
    {
        printf("\n~~~~~Stack underflow~~~~~");
        return -1;
    }
    item = s[top];
    top = top - 1;
    return item;
}
void display()
{
    int i;
    if (top == -1)
    {
        printf("\n~~~~~Stack is empty~~~~~");
        return;
    }
    printf("\nStack elements are:\n");
    for (i = top; i >= 0; i--)
        printf("| %d |\n", s[i]);
}
void palindrome()
{
    int flag = 1, i;
    printf("\nStack content are:\n");
```

```
for (i = top; i >= 0; i--)
    printf("| %d |", s[i]);
printf("\nReverse of stack content are:\n");
for (i = 0; i <= top; i++)
    printf("| %d |", s[i]);
for (i = 0; i <= top / 2; i++)
{
    if (s[i] != s[top - i])
    {
        flag = 0;
        break;
    }
}
if (flag == 1)
{
    printf("\nIt is palindrome number");
}
else
{
    printf("\nIt is not a palindrome number");
}
}
```

Output:

~~~~~Menu~~~~~ :

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **1**

**Enter an element to be pushed: 11**

~~~~~Menu~~~~~:

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **1**

Enter an element to be pushed: 12

~~~~~Menu~~~~~ :

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **1**

**Enter an element to be pushed: 13**

~~~~~Menu~~~~~ :

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **1**

Enter an element to be pushed: 14

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **1**

**Enter an element to be pushed: 15**

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **1**

Enter an element to be pushed: 16

~~~~~Stack overflow~~~~~

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **4**

Stack elements are:

| 15 |
| 14 |
| 13 |
| 12 |
| 11 |

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **2**

**Element popped is: 15**

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: **4**

Stack elements are:

| 14 |
| 13 |
| 12 |
| 11 |

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: 2

**Element popped is: 14**

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: 2

Element popped is: 13

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: 2

**Element popped is: 12**

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: 2

Element popped is: 11

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo
- =>2.Pop an Element from Stack and Underflow demo
- =>3.Palindrome demo
- =>4.Display
- =>5.Exit

Enter your choice: 2

~~~~~Stack underflow~~~~~

~~~~~Menu~~~~~

- =>1.Push an Element to Stack and Overflow demo

---

```
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 4
~~~~Stack is empty~~~~
~~~~Menu~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 11
~~~~Menu~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 22
~~~~Menu~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 11
~~~~Menu~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 3
Stack content are:
| 11 |
| 22 |
| 11 |
```

---

**Reverse of stack content are:**

| 11 |

| 22 |

| 11 |

**It is palindrome number**

~~~~~Menu~~~~~

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 2

**Element popped is: 11**

~~~~~Menu~~~~~

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 2

**Element popped is: 22**

~~~~~Menu~~~~~

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 2

**Element popped is: 11**

~~~~~Menu~~~~~

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1**Enter an element to be pushed: 11**

~~~~~Menu~~~~~

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

**Enter an element to be pushed: 22**

~~~~~Menu~~~~~

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

**Enter an element to be pushed: 33**

~~~~~Menu~~~~~

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 3

**Stack content are:**

| 33 |

| 22 |

| 11 |

**Reverse of stack content are:**

| 11 |

| 22 |

| 33 |

**It is not a palindrome number**

~~~~~Menu~~~~~

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 5

Program terminated successfully.

**3. Develop a program in C for simulating a Printer Queue using Queue data structure with the following operations:**

- a. Add a print job to the Printer Queue.**
- b. Process (delete) a print job from the Printer Queue.**
- c. Display all pending print jobs in the Queue.**
- d. Display the number of print jobs currently waiting in the Queue.**
- e. Demonstrate Queue overflow and underflow conditions.**
- f. Exit.**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Define the maximum size of the printer queue to easily demonstrate overflow
#define MAX 3
// Structure to represent a print job
typedef struct {
 int job_id;
 char document_name[50];
} PrintJob;
// Global queue variables
PrintJob printer_queue[MAX];
int front = -1;
int rear = -1;
// Function to check if the queue is full (Overflow condition)
int isFull()
{
 return (rear == MAX - 1);
}
// Function to check if the queue is empty (Underflow condition)
int isEmpty()
{
 return (front == -1 || front > rear);
}
// a. Add a print job to the Printer Queue (Enqueue)
void addJob()
{
 // e. Demonstrate Queue overflow condition
 if (isFull())
 {
 printf("\n● QUEUE OVERFLOW! The printer queue is full. Cannot add more print jobs.\n");
 return;
 }
 PrintJob new_job;
 printf("Enter Job ID (integer): ");
 scanf("%d", &new_job.job_id);
 getchar(); // Clear newline character from input buffer
 printf("Enter Document Name: ");
 fgets(new_job.document_name, sizeof(new_job.document_name), stdin);
 // Remove newline character
```

```
new_job.document_name[strcspn(new_job.document_name, "\n")] = 0;

if (front == -1)
{
 front = 0;
}
rear++;
printer_queue[rear] = new_job;
printf("✓ Success: Print Job [%d: %s] added to the queue.\n", new_job.job_id,
 new_job.document_name);
}
void processJob()
{
 if (isEmpty())
 {
 printf("\n⚠ QUEUE UNDERFLOW! No pending print jobs to process.\n");
 return;
 }
 PrintJob processed_job = printer_queue[front];
 printf("\n🖨 Processing Print Job...\n");
 printf("-> Job ID: %d\n", processed_job.job_id);
 printf(" -> Document: '%s' has been printed successfully!\n", processed_job.document_name);
 front++;
 if (front > rear)
 {
 front = -1;
 rear = -1;
 }
}
void displayQueue()
{
 int i;
 if (isEmpty())
 {
 printf("\nThe Printer Queue is currently empty.\n");
 return;
 }
 printf("\n--- PENDING PRINT JOBS IN QUEUE ---\n");
 printf("%-10s %-30s\n", "Job ID", "Document Name");
 printf("-----\n");
 for (i = front; i <= rear; i++)
 {
 printf("%-10d %-30s\n", printer_queue[i].job_id, printer_queue[i].document_name);
 }
 printf("-----\n");
}
void displayCount()
{

```

```
 if (isEmpty())
 {
 printf("\nNumber of print jobs currently waiting: 0\n");
 }
 else
 {
 int count = rear - front + 1;
 printf("\nNumber of print jobs currently waiting: %d\n",count);
 }
}
// Main Driver Function
int main()
{
 int choice;
 while (1)
 {
 printf("\n===== PRINTER QUEUE SIMULATION =====");
 printf("\n1. Add a print job");
 printf("\n2. Process (delete) a print job");
 printf("\n3. Display all pending print jobs");
 printf("\n4. Display waiting job count");
 printf("\n5. Exit");
 printf("\n=====");
 printf("\nEnter your choice (1-5): ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 addJob();
 break;
 case 2:
 processJob();
 break;
 case 3:
 displayQueue();
 break;
 case 4:
 displayCount();
 break;
 case 5:
 printf("\nExiting printer simulation. Goodbye!\n");
 exit(0);
 default:
 printf("\nX Invalid choice! Please select an option between 1 and 5.\n");
 }
 }
 return 0;
}
```

**Output:**

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 3

The Printer Queue is currently empty.

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 2

⚠ QUEUE UNDERFLOW! No pending print jobs to process.

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 1

Enter Job ID (integer): 101

Enter Document Name: Resume\_2026.pdf

✓ Success: Print Job [101: Resume\_2026.pdf] added to the queue.

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 1

Enter Job ID (integer): 102

Enter Document Name: Project\_Report.docx

✓ Success: Print Job [102: Project\_Report.docx] added to the queue.

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 1

Enter Job ID (integer): 103

Enter Document Name: Event\_Flyer.png

✓ Success: Print Job [103: Event\_Flyer.png] added to the queue.

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 4

Number of print jobs currently waiting: 3

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 1

● QUEUE OVERFLOW! The printer queue is full. Cannot add more print jobs.

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 3

--- PENDING PRINT JOBS IN QUEUE ---

Job ID    Document Name

-----

101      Resume\_2026.pdf

102      Project\_Report.docx

103      Event\_Flyer.png

---

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 2

 Processing Print Job...

-> Job ID: 101

-> Document: 'Resume\_2026.pdf' has been printed successfully!

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 4

Number of print jobs currently waiting: 2

===== PRINTER QUEUE SIMULATION =====

1. Add a print job
2. Process (delete) a print job
3. Display all pending print jobs
4. Display waiting job count
5. Exit

=====

Enter your choice (1-5): 5

Exiting printer simulation. Goodbye!

**4. Develop a menu-driven Program in C for the following operations on a Singly Linked****List of Student Records:**

- a. Create a linked list using front insertion.**
- b. Display all student records and count the number of nodes.**
- c. Insert and delete nodes at the beginning and end of the list.**
- d. Search for a student record using USN.**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Student
{
 char usn[15];
 char name[30];
 char branch[10];
 int sem;
 char phno[15];
 struct Student *next;
};
typedef struct Student *NODE;
NODE getnode()
{
 NODE temp = (NODE)malloc(sizeof(struct Student));
 if (temp == NULL)
 {
 printf("Memory allocation failed!\n");
 exit(0);
 }
 printf("Enter USN: ");
 scanf("%s", temp->usn);
 printf("Enter Name: ");
 scanf("%s", temp->name);
 printf("Enter Branch: ");
 scanf("%s", temp->branch);
 printf("Enter Semester: ");
 scanf("%d", &temp->sem);
 printf("Enter Phone Number: ");
 scanf("%s", temp->phno);
 temp->next = NULL;
 return temp;
}
NODE insert_front(NODE head)
{
 NODE temp = getnode();
 temp->next = head;
 printf("Node inserted at the front successfully.\n");
 return temp;
}
```

```
}
NODE insert_end(NODE head)
{
 NODE temp = getnode();
 if (head == NULL)
 {
 return temp;
 }
 NODE cur = head;
 while (cur->next != NULL)
 {
 cur = cur->next;
 }
 cur->next = temp;
 printf("Node inserted at the end successfully.\n");
 return head;
}
NODE delete_front(NODE head)
{
 if (head == NULL)
 {
 printf("List is empty! Deletion not possible.\n");
 return NULL;
 }
 NODE temp = head;
 head = head->next;
 printf("Deleted record with USN: %s\n", temp->usn);
 free(temp);
 return head;
}
NODE delete_end(NODE head)
{
 if (head == NULL)
 {
 printf("List is empty! Deletion not possible.\n");
 return NULL;
 }
 if (head->next == NULL)
 {
 printf("Deleted record with USN: %s\n", head->usn);
 free(head);
 return NULL;
 }
 NODE cur = head;
 NODE prev = NULL;
 while (cur->next != NULL)
 {
 prev = cur;
```

```
 cur = cur->next;
 }
 prev->next = NULL;
 printf("Deleted record with USN: %s\n", cur->usn);
 free(cur);
 return head;
}
void display(NODE head)
{
 if (head == NULL)
 {
 printf("The linked list is empty.\n");
 return;
 }
 NODE cur = head;
 int count = 0;
 printf("\n--- STUDENT RECORDS ---\n");

 while (cur != NULL)
 {
 printf("USN: %s | Name: %s | Branch: %s | Sem: %d | Phone: %s\n",
 cur->usn,
 cur->name,
 cur->branch,
 cur->sem,
 cur->phno);
 cur = cur->next;
 count++;
 }
 printf("Total number of nodes: %d\n\n", count);
}
void search_usn(NODE head)
{
 if (head == NULL)
 {
 printf("List is empty! Search not possible.\n");
 return;
 }
 char target[15];
 printf("Enter USN to search: ");
 scanf("%s", target);
 NODE cur = head;
 int pos = 1;
 while (cur != NULL)
 {
 if (strcmp(cur->usn, target) == 0)
 {
 printf("Record found at position %d:\n", pos);
 }
 }
}
```

```
 printf("USN: %s | Name: %s | Branch: %s | Sem: %d | Phone: %s\n",
 cur->usn,
 cur->name,
 cur->branch,
 cur->sem,
 cur->phno);
 return;
 }
 cur = cur->next;
 pos++;
}
printf("Record with USN %s not found.\n", target);
}
int main()
{
 NODE head = NULL;
 int choice, n, i;
 while (1)
 {
 printf("\n--- MENU ---\n");
 printf("1. Create list using front insertion (N students)\n");
 printf("2. Display records and count nodes\n");
 printf("3. Insert at front\n");
 printf("4. Insert at end\n");
 printf("5. Delete from front\n");
 printf("6. Delete from end\n");
 printf("7. Search by USN\n");
 printf("8. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice)
 {
 case 1:
 printf("Enter number of students: ");
 scanf("%d", &n);
 for (i = 0; i < n; i++)
 {
 head = insert_front(head);
 }
 break;
 case 2:
 display(head);
 break;
 case 3:
 head = insert_front(head);
 break;
 case 4:
 head = insert_end(head);
```

```
 break;
 case 5:
 head = delete_front(head);
 break;
 case 6:
 head = delete_end(head);
 break;
 case 7:
 search_usn(head);
 break;
 case 8:
 exit(0);
 default:
 printf("Invalid choice! Please try again.\n");
 }
}
return 0;
}
```

**Output:**

--- MENU ---

1. Create list using front insertion (N students)
2. Display records and count nodes
3. Insert at front
4. Insert at end
5. Delete from front
6. Delete from end
7. Search by USN
8. Exit

Enter your choice: 1

Enter number of students: 2

Enter USN: 1AY23CS002

Enter Name: Bob

Enter Branch: CSE

Enter Semester: 4

Enter Phone Number: 9876543210

Node inserted at the front successfully.

Enter USN: 1AY23CS001

Enter Name: Alice

Enter Branch: CSE

Enter Semester: 4

Enter Phone Number: 9123456780

Node inserted at the front successfully.

--- MENU ---

Enter your choice: 2

--- STUDENT RECORDS ---

USN: 1AY23CS001 | Name: Alice | Branch: CSE | Sem: 4 | Phone: 9123456780

USN: 1AY23CS002 | Name: Bob | Branch: CSE | Sem: 4 | Phone: 9876543210

Total number of nodes: 2

--- MENU ---

Enter your choice: 7

Enter USN to search: 1AY23CS002

Record found at position 2:

USN: 1AY23CS002 | Name: Bob | Branch: CSE | Sem: 4 | Phone: 9876543210

--- MENU ---

Enter your choice: 5

Deleted record with USN: 1AY23CS001

--- MENU ---

Enter your choice: 2

--- STUDENT RECORDS ---

USN: 1AY23CS002 | Name: Bob | Branch: CSE | Sem: 4 | Phone: 9876543210

Total number of nodes: 1

--- MENU ---

Enter your choice: 8

**5. Develop a Program in C for the following:**

- a. Create a Binary Tree by inserting integer elements entered by the user in level order.
- b. Implement the following functions:
  - i. createTree() – Create a binary tree using dynamic memory allocation.
  - ii. preorder() – Display the elements using Preorder Traversal.
  - iii. inorder() – Display the elements using Inorder Traversal.
  - iv. postorder() – Display the elements using Postorder Traversal.
- c. Provide a menu-driven interface with the following options:
  - i. Create Binary Tree
  - ii. Display Preorder Traversal
  - iii. Display Inorder Traversal
  - iv. Display Postorder Traversal
  - v. Exit

```
#include <stdio.h>
#include <stdlib.h>
// Structure for a Binary Tree Node
struct Node {
 int data;
 struct Node* left;
 struct Node* right;
};
struct QueueNode {
 struct Node* treeNode;
 struct QueueNode* next;
};
struct Queue {
 struct QueueNode *front, *rear;
};
// Function to create a new tree node
struct Node* createTreeNode(int data) {
 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode->data = data;
 newNode->left = NULL;
 newNode->right = NULL;
 return newNode;
}
// Queue operations for tracking insertion points
void enqueue(struct Queue* q, struct Node* tNode)
{
 struct QueueNode* temp = (struct QueueNode*)malloc(sizeof(struct QueueNode));
 temp->treeNode = tNode;
 temp->next = NULL;
 if (q->rear == NULL)
 {
 q->front = q->rear = temp;
 return;
 }
}
```

```
 q->rear->next = temp;
 q->rear = temp;
}
struct Node* dequeue(struct Queue* q)
{
 if (q->front == NULL)
 return NULL;
 struct QueueNode* temp = q->front;
 struct Node* tNode = temp->treeNode;
 q->front = q->front->next;
 if (q->front == NULL)
 q->rear = NULL;
 free(temp);
 return tNode;
}
// i. Function to create/insert elements into a binary tree in level order
struct Node* createTree(struct Node* root)
{
 int n, value;
 int i;
 printf("Enter the number of elements to insert: ");
 scanf("%d", &n);
 if (n <= 0)
 {
 printf("No elements to insert.\n");
 return root;
 }
 struct Queue q = {NULL, NULL};
 for (i = 0; i < n; i++)
 {
 printf("Enter element %d: ", i + 1);
 scanf("%d", &value);
 struct Node* newNode = createTreeNode(value);
 if (root == NULL)
 {
 root = newNode;
 }
 else
 {
 // Level order insertion using the queue front node pointer
 struct QueueNode* current = q.front;
 while (current != NULL)
 {
 if (current->treeNode->left == NULL)
 {
 current->treeNode->left = newNode;
 break;
 }
 }
 }
 }
}
```

```
 else if
 (current->treeNode->right == NULL)
 {
 current->treeNode->right = newNode;
 // Front node has both children now, it can be safely
 // dequeued later, but we manage our current iteration
 // manually or via standard dequeue.
 dequeue(&q);
 break;
 }
 current = current->next;
 }
}
enqueue(&q, newNode);
}
printf("Binary Tree created successfully in level order!\n");
return root;
}
// ii. Preorder Traversal (Root -> Left -> Right)
void preorder(struct Node* root)
{
 if (root != NULL)
 {
 printf("%d ", root->data);
 preorder(root->left);
 preorder(root->right);
 }
}
// iii. Inorder Traversal (Left -> Root -> Right)
void inorder(struct Node* root)
{
 if (root != NULL)
 {
 inorder(root->left);
 printf("%d ", root->data);
 inorder(root->right);
 }
}
// iv. Postorder Traversal (Left -> Right -> Root)
void postorder(struct Node* root)
{
 if (root != NULL)
 {
 postorder(root->left);
 postorder(root->right);
 printf("%d ", root->data);
 }
}
}
```

---

// Main execution function with a menu-driven interface

```
int main()
{
 struct Node* root = NULL;
 int choice;
 while (1)
 {
 printf("\n--- BINARY TREE MENU ---\n");
 printf("1. Create Binary Tree (Level Order)\n");
 printf("2. Display Preorder Traversal\n");
 printf("3. Display Inorder Traversal\n");
 printf("4. Display Postorder Traversal\n");
 printf("5. Exit\n");
 printf("Enter your choice (1-5): ");
 scanf("%d", &choice);
 switch (choice)
 {
 case 1:
 root = createTree(root);
 break;
 case 2:
 if (root == NULL)
 {
 printf("Tree is empty!\n");
 }
 else
 {
 printf("Preorder Traversal: ");
 preorder(root);
 printf("\n");
 }
 break;
 case 3:
 if (root == NULL)
 {
 printf("Tree is empty!\n");
 }
 else
 {
 printf("Inorder Traversal: ");
 inorder(root);
 printf("\n");
 }
 break;
 case 4:
 if (root == NULL)
 {
 printf("Tree is empty!\n");
 }
 else
 {
 printf("Postorder Traversal: ");
 postorder(root);
 printf("\n");
 }
 break;
 case 5:
 return 0;
 default:
 printf("Invalid choice!\n");
 }
 }
}
```

```
 }
 else
 {
 printf("Postorder Traversal: ");
 postorder(root);
 printf("\n");
 }
 break;
case 5:
 printf("Exiting program.\n");
 exit(0);
default:
 printf("Invalid choice! Please select an option between 1 and 5.\n");
}
}
return 0;
}
```

**Output:**

--- BINARY TREE MENU ---

1. Create Binary Tree (Level Order)
2. Display Preorder Traversal
3. Display Inorder Traversal
4. Display Postorder Traversal
5. Exit

Enter your choice (1-5): 1

Enter the number of elements to insert: 5

Enter element 1: 10

Enter element 2: 20

Enter element 3: 30

Enter element 4: 40

Enter element 5: 55

Binary Tree created successfully in level order!

--- BINARY TREE MENU ---

1. Create Binary Tree (Level Order)
2. Display Preorder Traversal
3. Display Inorder Traversal
4. Display Postorder Traversal
5. Exit

Enter your choice (1-5): 2

Preorder Traversal: 10 20 40 55 30

--- BINARY TREE MENU ---

1. Create Binary Tree (Level Order)
2. Display Preorder Traversal
3. Display Inorder Traversal
4. Display Postorder Traversal
5. Exit

Enter your choice (1-5): 3

Inorder Traversal: 40 20 55 10 30

--- BINARY TREE MENU ---

1. Create Binary Tree (Level Order)
2. Display Preorder Traversal
3. Display Inorder Traversal
4. Display Postorder Traversal
5. Exit

Enter your choice (1-5): 4

Postorder Traversal: 40 55 20 30 10

--- BINARY TREE MENU ---

1. Create Binary Tree (Level Order)
2. Display Preorder Traversal
3. Display Inorder Traversal
4. Display Postorder Traversal
5. Exit

Enter your choice (1-5): 5

Exiting program.

**6. Develop a Program in C for the following operations on Graph(G) of Cities****a. Create an Adjacency Matrix representing the graph of N cities****b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method**

```
#include <stdio.h>
#include <stdlib.h>
int a[50][50], n, visited[50];
int q[50];
int front = -1, rear = -1;
/* BFS function */
void bfs(int v)
{
 int i, cur;
 visited[v] = 1;
 q[++rear] = v;

 while (front != rear)
 {
 cur = q[++front];

 for (i = 1; i <= n; i++)
 {
 if (a[cur][i] == 1 && visited[i] == 0)
 {
 q[++rear] = i;
 visited[i] = 1;
 printf("%d ", i);
 }
 }
 }
}
/* DFS function */
void dfs(int v)
{
 int i;
 visited[v] = 1;
 for (i = 1; i <= n; i++)
 {
 if (a[v][i] == 1 && visited[i] == 0)
 {
 printf("%d ", i);
 dfs(i);
 }
 }
}
int main()
{
 int ch, start, i, j;
```

```
/* Read number of vertices */
printf("\nEnter the number of vertices in graph: ");
scanf("%d", &n);
/* Read adjacency matrix */
printf("\nEnter the adjacency matrix:\n");
for (i = 1; i <= n; i++)
{
 for (j = 1; j <= n; j++)
 {
 scanf("%d", &a[i][j]);
 }
}
/* Read starting vertex */
printf("\nEnter the starting vertex: ");
scanf("%d", &start);

/* Menu loop */
while (1)
{
 /*
 * Reset visited array before every BFS/DFS.
 * Otherwise, the previous traversal will affect
 * the next traversal.
 */
 for (i = 1; i <= n; i++)
 {
 visited[i] = 0;
 }
 /* Reset BFS queue */
 front = -1;
 rear = -1;
 /* Display menu */
 printf("\n\n=====");
 printf("\n1. BFS: Print all nodes reachable from starting node");
 printf("\n2. DFS: Print all nodes reachable from starting node");
 printf("\n3. Exit");
 printf("\n\n=====");
 printf("\nEnter your choice: ");
 scanf("%d", &ch);
 switch (ch)
 {
 case 1:
 printf("\nNodes reachable from starting vertex %d are: ", start);
 bfs(start);
 /*
 * Check whether any vertices are unreachable
 */
 for (i = 1; i <= n; i++)
```

```
{
 if (visited[i] == 0)
 {
 printf("\nThe vertex that is not reachable is %d",
 i);
 }
}

break;
case 2:
 printf("\nNodes reachable from starting vertex %d are: ",
 start);
 dfs(start);

 /*
 * Check whether any vertices are unreachable
 */
 for (i = 1; i <= n; i++)
 {
 if (visited[i] == 0)
 {
 printf("\nThe vertex that is not reachable is %d", i);
 }
 }
 break;
case 3:
 printf("\nExiting program...\n");
 exit(0);
default:
 printf("\nPlease enter a valid choice.");
}
}
return 0;
}
```

## Output:

Enter the number of vertices in graph: 4

Enter the adjacency matrix:

0 1 1 0

1 0 0 1

1 0 0 1

0 1 1 0

Enter the starting vertex: 1

=====

1. BFS: Print all nodes reachable from starting node

2. DFS: Print all nodes reachable from starting node

3. Exit

=====

Enter your choice: 1

Nodes reachable from starting vertex 1 are: 2 3 4

=====

1. BFS: Print all nodes reachable from starting node

2. DFS: Print all nodes reachable from starting node

3. Exit

=====

Enter your choice: 2

Nodes reachable from starting vertex 1 are: 2 4 3

=====

1. BFS: Print all nodes reachable from starting node

2. DFS: Print all nodes reachable from starting node

3. Exit

=====

Enter your choice: 3

Exiting program...

Process returned 0 (0x0) execution time : 98.145 s

Press any key to continue.

---

## Part B

1. Develop a C program to:
  - a. Read two sparse matrices of the same order.
  - b. Convert both into 3-tuple representation.
  - c. Perform matrix addition.
  - d. Display the resultant matrix in both normal matrix form and 3-tuple form.

```
#include <stdio.h>
#define MAX 100
/* Function declarations */
void readMatrix(int matrix[MAX][MAX], int rows, int cols);
int convertToTriplet(
 int matrix[MAX][MAX],
 int rows,
 int cols,
 int triplet[MAX][3]
);
void addTriplets(
 int t1[MAX][3],
 int t2[MAX][3],
 int result[MAX][3]
);
void displayTriplet(int triplet[MAX][3]);
void displayNormalFromTriplet(int triplet[MAX][3]);
/* Main function */
int main()
{
 int rows, cols;
 int matrixA[MAX][MAX];
 int matrixB[MAX][MAX];
 int tripletA[MAX][3];
 int tripletB[MAX][3];
 int tripletResult[MAX][3];
 /* -----
 a. Read two sparse matrices
 ----- */
 printf("Enter the number of rows and columns: ");
 scanf("%d %d", &rows, &cols);
 printf("\nEnter elements of Matrix A:\n");
 readMatrix(matrixA, rows, cols);
 printf("\nEnter elements of Matrix B:\n");
 readMatrix(matrixB, rows, cols);
 /* -----
 b. Convert matrices to 3-tuple form
 ----- */
 convertToTriplet(matrixA, rows, cols, tripletA);
```

```

convertToTriplet(matrixB, rows, cols, tripletB);
printf("\n=====");
printf("\n3-TUPLE REPRESENTATION");
printf("\n=====");
printf("\n\nTriplet Representation of Matrix A:\n");
displayTriplet(tripletA);
printf("\nTriplet Representation of Matrix B:\n");
displayTriplet(tripletB);
/* -----
 c. Perform matrix addition
 ----- */
addTriplets(tripletA, tripletB, tripletResult);
/* -----
 d. Display resultant matrix
 in both forms
 ----- */
printf("\n=====");
printf("\nRESULTANT MATRIX (A + B)");
printf("\n=====");
/* Result in 3-tuple form */
printf("\n\n1. Resultant Matrix in 3-Tuple Form:\n");
displayTriplet(tripletResult);
/* Result in normal matrix form */
printf("\n\n2. Resultant Matrix in Normal Matrix Form:\n");
displayNormalFromTriplet(tripletResult);
return 0;
}
/* -----
 Function to read a matrix
 ----- */
void readMatrix(int matrix[MAX][MAX], int rows, int cols)
{
 int i, j;
 for (i = 0; i < rows; i++)
 {
 for (j = 0; j < cols; j++)
 {
 scanf("%d", &matrix[i][j]);
 }
 }
}
/* -----
 Function to convert normal matrix
 into 3-tuple representation
 ----- */
int convertToTriplet(
 int matrix[MAX][MAX],
 int rows,

```

```
 int cols,
 int triplet[MAX][3]
)
{
 int i, j;
 int k = 1;
 for (i = 0; i < rows; i++)
 {
 for (j = 0; j < cols; j++)
 {
 if (matrix[i][j] != 0)
 {
 triplet[k][0] = i;
 triplet[k][1] = j;
 triplet[k][2] = matrix[i][j];
 k++;
 }
 }
 }
 /* Header row:
 rows, columns, number of non-zero elements */
 triplet[0][0] = rows;
 triplet[0][1] = cols;
 triplet[0][2] = k - 1;
 return k - 1;
}
/* -----
 Function to add two triplet matrices
 ----- */
void addTriplets(
 int t1[MAX][3],
 int t2[MAX][3],
 int result[MAX][3]
)
{
 int i = 1;
 int j = 1;
 int k = 1;
 int sizeA = t1[0][2];
 int sizeB = t2[0][2];
 /* Store rows and columns */
 result[0][0] = t1[0][0];
 result[0][1] = t1[0][1];
 while (i <= sizeA && j <= sizeB)
 {
 /* Row of A is smaller */
 if (t1[i][0] < t2[j][0])
 {
 result[k][0] = t1[i][0];
 result[k][1] = t1[i][1];
 result[k][2] = t1[i][2];
 i++;
 k++;
 }
 else if (t2[j][0] < t1[i][0])
 {
 result[k][0] = t2[j][0];
 result[k][1] = t2[j][1];
 result[k][2] = t2[j][2];
 j++;
 k++;
 }
 else
 {
 result[k][0] = t1[i][0];
 result[k][1] = t1[i][1];
 result[k][2] = t1[i][2] + t2[j][2];
 i++;
 j++;
 k++;
 }
 }
 result[k][0] = i;
 result[k][1] = j;
 result[k][2] = k - 1;
 return k - 1;
}
```

```
 result[k][0] = t1[i][0];
 result[k][1] = t1[i][1];
 result[k][2] = t1[i][2];
 i++;
 k++;
 }
 /* Row of B is smaller */
 else if (t1[i][0] > t2[j][0])
 {
 result[k][0] = t2[j][0];
 result[k][1] = t2[j][1];
 result[k][2] = t2[j][2];
 j++;
 k++;
 }
 /* Same row */
 else
 {
 /* Column of A is smaller */
 if (t1[i][1] < t2[j][1])
 {
 result[k][0] = t1[i][0];
 result[k][1] = t1[i][1];
 result[k][2] = t1[i][2];
 i++;
 k++;
 }
 /* Column of B is smaller */
 else if (t1[i][1] > t2[j][1])
 {
 result[k][0] = t2[j][0];
 result[k][1] = t2[j][1];
 result[k][2] = t2[j][2];
 j++;
 k++;
 }
 /* Same row and same column */
 else
 {
 int sum;
 sum = t1[i][2] + t2[j][2];
 /* Store only non-zero result */
 if (sum != 0)
 {
 result[k][0] = t1[i][0];
 result[k][1] = t1[i][1];
 result[k][2] = sum;
 k++;
 }
 }
 }
}
```

```

 }
 i++;
 j++;
 }
}
/* Copy remaining elements of A */
while (i <= sizeA)
{
 result[k][0] = t1[i][0];
 result[k][1] = t1[i][1];
 result[k][2] = t1[i][2];
 i++;
 k++;
}
/* Copy remaining elements of B */
while (j <= sizeB)
{
 result[k][0] = t2[j][0];
 result[k][1] = t2[j][1];
 result[k][2] = t2[j][2];
 j++;
 k++;
}
/* Store number of non-zero elements */
result[0][2] = k - 1;
}
/* -----
Function to display 3-tuple
----- */
void displayTriplet(int triplet[MAX][3])
{
 int i;
 printf("\nRow\tColumn\tValue\n");
 printf("-----\n");
 for (i = 0; i <= triplet[0][2]; i++)
 {
 printf(
 "%d\t%d\t%d\n",
 triplet[i][0],
 triplet[i][1],
 triplet[i][2]
);
 }
}
/* -----
Function to display normal matrix
from triplet representation

```

```
----- */
void displayNormalFromTriplet(int triplet[MAX][3])
{
 int rows = triplet[0][0];
 int cols = triplet[0][1];
 int nonZero = triplet[0][2];
 int i, j;
 int k = 1;
 for (i = 0; i < rows; i++)
 {
 for (j = 0; j < cols; j++)
 {
 if (k <= nonZero &&
 triplet[k][0] == i &&
 triplet[k][1] == j)
 {
 printf("%d\t", triplet[k][2]);
 k++;
 }
 else
 {
 printf("0\t");
 }
 }
 printf("\n");
 }
}
```

**Output:**

Enter the number of rows and columns: 3 3

Enter elements of Matrix A:

1 0 2

0 3 0

4 0 5

Enter elements of Matrix B:

0 6 2

7 0 0

4 0 1

=====

### 3-TUPLE REPRESENTATION

=====

Triplet Representation of Matrix A:

Row    Column   Value

-----

3    3    5

0    0    1

0    2    2

1    1    3

2    0    4

2    2    5

Triplet Representation of Matrix B:

Row    Column   Value

3    3    5

0    1    6

0    2    2

1    0    7

2    0    4

2    2    1

=====

### RESULTANT MATRIX (A + B)

=====

1. Resultant Matrix in 3-Tuple Form:

Row    Column   Value

3    3    7

0    0    1

0    1    6

0    2    4

1    0    7

1    1    3

2    0    8

2    2    6

2. Resultant Matrix in Normal Matrix Form:

1    6    4

7    3    0

8    0    6

**2. Develop a C-based Expression Conversion tool that uses stack data structure to convert arithmetic expressions from infix notation to postfix notation. The tool should support alphanumeric operands, parentheses, and the operators (+, -, \*, /, %, ^). Analyze the correctness of the conversion by testing various valid expressions.**

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
typedef struct
{
 char data[MAX];
 int top;
} Stack;
void initStack(Stack *s)
{
 s->top = -1;
}
int isEmpty(Stack *s)
{
 return s->top == -1;
}
void push(Stack *s, char ch)
{
 if (s->top == MAX - 1)
 {
 printf("Error: Stack Overflow!\n");
 exit(1);
 }
 s->data[++s->top] = ch;
}
// Pop element from stack
char pop(Stack *s)
{
 if (isEmpty(s))
 {
 return '\0';
 }
 return s->data[s->top--];
}
char peek(Stack *s)
{
 if (isEmpty(s))
 {
 return '\0';
 }
}
```

```
 return s->data[s->top];
}
int isOperator(char ch)
{
 return (ch == '+' || ch == '-' ||
 ch == '*' || ch == '/' ||
 ch == '%' || ch == '^');
}
int precedence(char ch)
{
 switch (ch)
 {
 case '^':
 return 3;
 case '*':
 case '/':
 case '%':
 return 2;
 case '+':
 case '-':
 return 1;
 default:
 return 0;
 }
}
int isRightAssociative(char ch)
{
 return ch == '^';
}
void infixToPostfix(char infix[], char postfix[])
{
 Stack s;
 int i = 0;
 int j = 0;
 char ch;
 initStack(&s);
 while (infix[i] != '\0')
 {
 ch = infix[i];
 if (isspace((unsigned char)ch))
 {
 i++;
 continue;
 }
 if (isalnum((unsigned char)ch))
 {
 postfix[j++] = ch;
 }
 }
}
```

```
// If opening parenthesis, push it
else if (ch == '(')
{
 push(&s, ch);
}
// If closing parenthesis, pop until '('
else if (ch == ')')
{
 while (!isEmpty(&s) && peek(&s) != '(')
 {
 postfix[j++] = pop(&s);
 }
 if (!isEmpty(&s) && peek(&s) == '(')
 {
 pop(&s);
 }
}
// If operator
else if (isOperator(ch))
{
 while (!isEmpty(&s) &&
 peek(&s) != '(' &&
 (precedence(peek(&s)) > precedence(ch) ||
 (precedence(peek(&s)) == precedence(ch) &&
 !isRightAssociative(ch))))
 {
 postfix[j++] = pop(&s);
 }
 push(&s, ch);
}
i++;
}
// Pop remaining operators
while (!isEmpty(&s))
{
 postfix[j++] = pop(&s);
}
postfix[j] = '\0';
}
// Test several expressions
void testExpressions()
{
 char expressions[][MAX] = {
 "A+B",
 "A+B*C",
 "(A+B)*C",
 "A*(B+C)",
 "A+B*C-D",
 }
```

```

 "A*(B+C)/D",
 "A+B*(C-D)",
 "(A+B)*(C-D)",
 "A^B^C",
 "A%B+C*D",
 "((A+B)*C-D)/E",
 "A+B*(C^D-E)^(F+G*H)-I"
};
char expected[][MAX] = {
 "AB+",
 "ABC*+",
 "AB+C*",
 "ABC+*",
 "ABC*+D-",
 "ABC+*D/",
 "ABCD-*+",
 "AB+CD-*",
 "ABC^^",
 "AB%CD*+",
 "AB+C*D-E/",
 "ABCD^E-FGH*+^*+I-"
};
char postfix[MAX];
int i;
int total = sizeof(expressions) / sizeof(expressions[0]);
int passed = 0;
printf("\n=====\\n");
printf(" EXPRESSION TEST RESULTS\\n");
printf("=====\\n");
printf("%-4s %-35s %-25s %-10s\\n",
 "No.", "Infix Expression", "Postfix Expression", "Result");
printf("-----\\n");
for (i = 0; i < total; i++)
{
 infixToPostfix(expressions[i], postfix);

 if (strcmp(postfix, expected[i]) == 0)
 {
 printf("%-4d %-35s %-25s %-10s\\n",
 i + 1, expressions[i], postfix, "PASS");
 passed++;
 }
 else
 {
 printf("%-4d %-35s %-25s %-10s\\n",
 i + 1, expressions[i], postfix, "FAIL");
 }
}
}

```

```
printf("-----\n");
printf("Tests Passed : %d/%d\n", passed, total);

if (passed == total)
{
 printf("Status : ALL TESTS PASSED\n");
}
else
{
 printf("Status : SOME TESTS FAILED\n");
}
printf("=====\n");
}
int main()
{
 int choice;
 char infix[MAX];
 char postfix[MAX];
 while (1)
 {
 printf("\n=====\n");
 printf(" EXPRESSION CONVERSION TOOL\n");
 printf("=====\n");
 printf("1. Convert Infix to Postfix\n");
 printf("2. Test Various Valid Expressions\n");
 printf("3. Exit\n");
 printf("=====\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 getchar(); // Remove newline
 switch (choice)
 {
 case 1:
 printf("\nEnter an infix expression: ");
 fgets(infix, MAX, stdin);
 infix[strcspn(infix, "\n")] = '\0';
 infixToPostfix(infix, postfix);
 printf("\n----- RESULT ----- \n");
 printf("Infix Expression : %s\n", infix);
 printf("Postfix Expression : %s\n", postfix);
 printf("-----\n");
 break;
 case 2:
 testExpressions();
 break;
 case 3:
 printf("\nThank you for using Expression Conversion Tool!\n");
 return 0;
 }
 }
}
```

```
 default:
 printf("\nInvalid choice! Please enter 1, 2, or 3.\n");
 }
 }
 return 0;
}
```

**Output:**

## ===== EXPRESSION CONVERSION TOOL =====

1. Convert Infix to Postfix
2. Test Various Valid Expressions
3. Exit

=====
Enter your choice: 1

Enter an infix expression: A+B\*(C-D)

----- RESULT -----

Infix Expression: A+B\*(C-D)

Postfix Expression: ABCD-\*+
=====

## ===== EXPRESSION CONVERSION TOOL =====

1. Convert Infix to Postfix
2. Test Various Valid Expressions
3. Exit

=====
Enter your choice: 2
=====

## ===== EXPRESSION TEST RESULTS =====

| No. | Infix Expression      | Postfix Expression | Result |
|-----|-----------------------|--------------------|--------|
| 1   | A+B                   | AB+                | PASS   |
| 2   | A+B*C                 | ABC*+              | PASS   |
| 3   | (A+B)*C               | AB+C*              | PASS   |
| 4   | A*(B+C)               | ABC+*              | PASS   |
| 5   | A+B*C-D               | ABC*+D-            | PASS   |
| 6   | A*(B+C)/D             | ABC+*D/            | PASS   |
| 7   | A+B*(C-D)             | ABCD-*+            | PASS   |
| 8   | (A+B)*(C-D)           | AB+CD-*            | PASS   |
| 9   | A^B^C                 | ABC^^              | PASS   |
| 10  | A%B+C*D               | AB%CD*+            | PASS   |
| 11  | ((A+B)*C-D)/E         | AB+C*D-E/          | PASS   |
| 12  | A+B*(C^D-E)^(F+G*H)-I | ABCD^E-FGH*+^*+I-  | PASS   |

Tests Passed:12/12

Status: ALL TESTS PASSED

=====

=====

EXPRESSION CONVERSION TOOL

=====

1. Convert Infix to Postfix
  2. Test Various Valid Expressions
  3. Exit
- =====

Enter your choice: 3

Thank you for using Expression Conversion Tool!

Process returned 0 (0x0) execution time: 635.908 s

Press any key to continue.

**3. Develop a menu-driven Program in C for the following operations on Circular Queue (integer/floating point data):**

- a. Insert an element into the Circular Queue.
- b. Delete an element from the Circular Queue.
- c. Display the contents of the Circular Queue.
- d. Demonstrate overflow and underflow conditions.
- e. Exit.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int queue[MAX];
int front = -1;
int rear = -1;
int isEmpty()
{
 return (front == -1);
}
int isFull()
{
 return ((rear + 1) % MAX == front);
}
void insert()
{
 int item;
 if (isFull())
 {
 printf("\n*** OVERFLOW ***\n");
 printf("Circular Queue is FULL. Cannot insert an element.\n");
 return;
 }
 printf("\nEnter the element to insert: ");
 scanf("%d", &item);
 if (front == -1)
 {
 front = 0;
 }
 rear = (rear + 1) % MAX;
 queue[rear] = item;
 printf("%d inserted successfully.\n", item);
}
void deleteElement()
{
 int item;
 if (isEmpty())
 {
 printf("\n*** UNDERFLOW ***\n");
 printf("Circular Queue is EMPTY. Cannot delete an element.\n");
 }
}
```

```
 return;
 }
 item = queue[front];
 if (front == rear)
 {
 front = -1;
 rear = -1;
 }
 else
 {
 front = (front + 1) % MAX;
 }
 printf("%d deleted successfully.\n", item);
}

void display()
{
 int i;
 if (isEmpty())
 {
 printf("\nCircular Queue is EMPTY.\n");
 return;
 }
 printf("\nCircular Queue contents:\n");
 printf("Front -> ");
 i = front;
 while (1)
 {
 printf("%d ", queue[i]);

 if (i == rear)
 break;
 i = (i + 1) % MAX;
 }
 printf("<- Rear\n");
}

void demonstrate()
{
 int i;
 printf("\n===== \n");
 printf(" OVERFLOW AND UNDERFLOW DEMONSTRATION\n");
 printf("===== \n");
 front = -1;
 rear = -1;
 printf("\n1. UNDERFLOW Demonstration\n");
 printf("----- \n");
 printf("Trying to delete from an empty queue...\n");
 deleteElement();
 printf("\n2. OVERFLOW Demonstration\n");
}
```

```
printf("-----\n");
printf("Inserting %d elements into the queue...\n", MAX);
for (i = 1; i <= MAX; i++)
{
 if (!isFull())
 {
 if (front == -1)
 front = 0;
 rear = (rear + 1) % MAX;
 queue[rear] = i * 10;
 printf("Inserted: %d\n", queue[rear]);
 }
}
display();
printf("\nTrying to insert one more element...\n");
insert();
printf("\n===== \n");
printf(" Demonstration Completed\n");
printf("===== \n");
}
int main()
{
 char choice;
 while (1)
 {
 printf("\n\n===== \n");
 printf(" CIRCULAR QUEUE MENU\n");
 printf("===== \n");
 printf("e. Insert an element\n");
 printf("f. Delete an element\n");
 printf("g. Display the Circular Queue\n");
 printf("h. Demonstrate Overflow and Underflow\n");
 printf("i. Exit\n");
 printf("===== \n");
 printf("Enter your choice: ");
 scanf(" %c", &choice);
 switch (choice)
 {
 case 'e':
 case 'E':
 insert();
 break;
 case 'f':
 case 'F':
 deleteElement();
 break;
 case 'g':
 case 'G':
```

```
 display();
 break;
 case 'h':
 case 'H':
 demonstrate();
 break;
 case 'i':
 case 'I':
 printf("\nProgram exited successfully.\n");
 exit(0);
 default:
 printf("\nInvalid choice!\n");
 printf("Please enter e, f, g, h, or i.\n");
 }
}
return 0;
}
```

**Output:**

```
=====
 CIRCULAR QUEUE MENU
=====
e. Insert an element
f. Delete an element
g. Display the Circular Queue
h. Demonstrate Overflow and Underflow
i. Exit
=====
Enter your choice: e
Enter the element to insert: 10
10 inserted successfully.
=====
 CIRCULAR QUEUE MENU
=====
e. Insert an element
f. Delete an element
g. Display the Circular Queue
h. Demonstrate Overflow and Underflow
i. Exit
=====
Enter your choice: e
Enter the element to insert: 20
20 inserted successfully.
=====
 CIRCULAR QUEUE MENU
=====
e. Insert an element
f. Delete an element
g. Display the Circular Queue
h. Demonstrate Overflow and Underflow
i. Exit
=====
Enter your choice: e
Enter the element to insert: 30
30 inserted successfully.
=====
 CIRCULAR QUEUE MENU
=====
e. Insert an element
```

- f. Delete an element
- g. Display the Circular Queue
- h. Demonstrate Overflow and Underflow
- i. Exit

=====

Enter your choice: e

Enter the element to insert: 40

40 inserted successfully.

=====

#### CIRCULAR QUEUE MENU

=====

- e. Insert an element
- f. Delete an element
- g. Display the Circular Queue
- h. Demonstrate Overflow and Underflow
- i. Exit

=====

Enter your choice: g

Circular Queue contents:

Front -> 10 20 30 40 <- Rear

=====

#### CIRCULAR QUEUE MENU

=====

- e. Insert an element
- f. Delete an element
- g. Display the Circular Queue
- h. Demonstrate Overflow and Underflow
- i. Exit

=====

Enter your choice: f

10 deleted successfully.

=====

#### CIRCULAR QUEUE MENU

=====

- e. Insert an element
- f. Delete an element
- g. Display the Circular Queue
- h. Demonstrate Overflow and Underflow
- i. Exit

=====

Enter your choice: f

20 deleted successfully.

=====

CIRCULAR QUEUE MENU

=====

e. Insert an element

f. Delete an element

g. Display the Circular Queue

h. Demonstrate Overflow and Underflow

i. Exit

=====

Enter your choice: f

30 deleted successfully.

=====

CIRCULAR QUEUE MENU

=====

e. Insert an element

f. Delete an element

g. Display the Circular Queue

h. Demonstrate Overflow and Underflow

i. Exit

=====

Enter your choice: f

40 deleted successfully.

=====

CIRCULAR QUEUE MENU

=====

e. Insert an element

f. Delete an element

g. Display the Circular Queue

h. Demonstrate Overflow and Underflow

i. Exit

=====

Enter your choice: g

Circular Queue is EMPTY.

=====

CIRCULAR QUEUE MENU

=====

e. Insert an element

f. Delete an element

g. Display the Circular Queue

h. Demonstrate Overflow and Underflow

i. Exit

Enter your choice: h

### OVERFLOW AND UNDERFLOW DEMONSTRATION

#### 1. UNDERFLOW Demonstration

Trying to delete from an empty queue...

\*\*\* UNDERFLOW \*\*\*

Circular Queue is EMPTY. Cannot delete an element.

#### 2. OVERFLOW Demonstration

Inserting 5 elements into the queue...

Inserted: 10

Inserted: 20

Inserted: 30

Inserted: 40

Inserted: 50

Circular Queue contents:

Front -> 10 20 30 40 50 <- Rear

Trying to insert one more element...

\*\*\* OVERFLOW \*\*\*

Circular Queue is FULL. Cannot insert an element.

Demonstration Completed

### CIRCULAR QUEUE MENU

e. Insert an element

f. Delete an element

g. Display the Circular Queue

h. Demonstrate Overflow and Underflow

i. Exit

Enter your choice: i

Program exited successfully.

Process returned 0 (0x0) execution time : 43.887 s

Press any key to continue.

**4. Develop a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo etc.**

- a. Create a DLL of N Employees Data by using end insertion.**
- b. Display the status of DLL and count the number of nodes in it**
- c. Perform Insertion and Deletion at End of DLL**
- d. Perform Insertion and Deletion at Front of DLL**
- e. Demonstrate how this DLL can be used as Double Ended Queue.**
- f. Exit**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Employee
{
 char ssn[20];
 char name[50];
 char dept[30];
 char designation[30];
 float sal;
 char phNo[20];
 struct Employee *prev;
 struct Employee *next;
};
typedef struct Employee Node;
Node *head = NULL;
Node *tail = NULL;
Node *createNode()
{
 Node *newNode;
 newNode = (Node *)malloc(sizeof(Node));
 if (newNode == NULL)
 {
 printf("\nMemory allocation failed!\n");
 exit(1);
 }
 printf("\nEnter Employee SSN : ");
 scanf(" %19[^\n]", newNode->ssn);
 printf("Enter Employee Name : ");
 scanf(" %49[^\n]", newNode->name);
 printf("Enter Department : ");
 scanf(" %29[^\n]", newNode->dept);
 printf("Enter Designation : ");
 scanf(" %29[^\n]", newNode->designation);
 printf("Enter Salary : ");
 scanf("%f", &newNode->sal);
 printf("Enter Phone Number : ");
 scanf(" %19[^\n]", newNode->phNo);
 newNode->prev = NULL;
```

```
newNode->next = NULL;
return newNode;
}
void insertEnd()
{
 Node *newNode = createNode();
 if (head == NULL)
 {
 head = tail = newNode;
 }
 else
 {
 newNode->prev = tail;
 tail->next = newNode;
 tail = newNode;
 }
 printf("\nEmployee inserted at the END successfully.\n");
}
void insertFront()
{
 Node *newNode = createNode();
 if (head == NULL)
 {
 head = tail = newNode;
 }
 else
 {
 newNode->next = head;
 head->prev = newNode;
 head = newNode;
 }
 printf("\nEmployee inserted at the FRONT successfully.\n");
}
void deleteEnd()
{
 Node *temp;
 if (head == NULL)
 {
 printf("\nDLL is EMPTY. Nothing to delete.\n");
 return;
 }
 temp = tail;

 if (head == tail)
 {
 head = tail = NULL;
 }
 else
```

```
{
 tail = tail->prev;
 tail->next = NULL;
}
printf("\nEmployee deleted from END:\n");
printf("SSN : %s\n", temp->ssn);
printf("Name: %s\n", temp->name);
free(temp);
}
void deleteFront()
{
 Node *temp;
 if (head == NULL)
 {
 printf("\nDLL is EMPTY. Nothing to delete.\n");
 return;
 }
 temp = head;

 if (head == tail)
 {
 head = tail = NULL;
 }
 else
 {
 head = head->next;
 head->prev = NULL;
 }
 printf("\nEmployee deleted from FRONT:\n");
 printf("SSN : %s\n", temp->ssn);
 printf("Name: %s\n", temp->name);
 free(temp);
}
void display()
{
 Node *temp;
 int count = 0;

 if (head == NULL)
 {
 printf("\nDLL is EMPTY.\n");
 printf("Number of nodes = 0\n");
 return;
 }
 temp = head;
 printf("\n=====");
 printf("EMPLOYEE DLL\n");
 printf("=====");
}
```

```
printf("%-15s %-20s %-15s %-18s %-12s %-15s\n",
 "SSN", "Name", "Department",
 "Designation", "Salary", "Phone No.");
printf("-----\n");
while (temp != NULL)
{
 printf("%-15s %-20s %-15s %-18s %-12.2f %-15s\n",
 temp->ssn,
 temp->name,
 temp->dept,
 temp->designation,
 temp->sal,
 temp->phNo);
 count++;
 temp = temp->next;
}
printf("-----\n");
printf("Number of nodes = %d\n", count);
printf("=====\n");
}
void displayReverse()
{
 Node *temp;

 if (tail == NULL)
 {
 printf("\nDLL is EMPTY.\n");
 return;
 }
 temp = tail;
 printf("\nDLL in Reverse Order:\n");
 printf("-----\n");
 while (temp != NULL)
 {
 printf("SSN: %-15s | Name: %-20s | Dept: %-15s\n",
 temp->ssn, temp->name, temp->dept);
 temp = temp->prev;
 }
 printf("-----\n");
}
void createDLL()
{
 int n, i;
 printf("\nEnter number of employees: ");
 scanf("%d", &n);
 if (n <= 0)
 {
 printf("\nInvalid number of employees!\n");
 }
}
```

```
 return;
 }
 while (head != NULL)
 {
 deleteFront();
 }
 for (i = 1; i <= n; i++)
 {
 printf("\n===== Enter Employee %d Details =====\n", i);
 insertEnd();
 }
 printf("\nDLL with %d employees created successfully.\n", n);
}
void demonstrateDeque()
{
 int choice;
 while (1)
 {
 printf("\n===== \n");
 printf(" DOUBLY LINKED LIST AS DEQUE\n");
 printf("===== \n");
 printf("1. Insert at Front\n");
 printf("2. Insert at Rear\n");
 printf("3. Delete from Front\n");
 printf("4. Delete from Rear\n");
 printf("5. Display Deque\n");
 printf("6. Return to Main Menu\n");
 printf("===== \n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice)
 {
 case 1:
 insertFront();
 break;
 case 2:
 insertEnd();
 break;
 case 3:
 deleteFront();
 break;
 case 4:
 deleteEnd();
 break;
 case 5:
 display();
 break;
 case 6:
```

```
 return;
 default:
 printf("\nInvalid choice!\n");
 }
}
}
void freeList()
{
 Node *temp;
 while (head != NULL)
 {
 temp = head;
 head = head->next;
 free(temp);
 }
 tail = NULL;
}
int main()
{
 int choice;
 while (1)
 {
 printf("\n\n=====\\n");
 printf(" DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT\\n");
 printf("=====\\n");
 printf("1. Create DLL of N Employees using End Insertion\\n");
 printf("2. Display DLL and Count Number of Nodes\\n");
 printf("3. Insert Employee at End\\n");
 printf("4. Delete Employee from End\\n");
 printf("5. Insert Employee at Front\\n");
 printf("6. Delete Employee from Front\\n");
 printf("7. Demonstrate DLL as Double Ended Queue\\n");
 printf("8. Exit\\n");
 printf("=====\\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice)
 {
 case 1:
 createDLL();
 break;
 case 2:
 display();
 break;
 case 3:
 insertEnd();
 break;
 case 4:
```

```
 deleteEnd();
 break;
 case 5:
 insertFront();
 break;
 case 6:
 deleteFront();
 break;
 case 7:
 demonstrateDeque();
 break;
 case 8:
 freeList();
 printf("\nProgram terminated successfully.\n");
 return 0;

 default:
 printf("\nInvalid choice! Please try again.\n");
 }
}
return 0;
}
```

**Output:**

## =====

## DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

- =====
1. Create DLL of N Employees using End Insertion
  2. Display DLL and Count Number of Nodes
  3. Insert Employee at End
  4. Delete Employee from End
  5. Insert Employee at Front
  6. Delete Employee from Front
  7. Demonstrate DLL as Double Ended Queue
  8. Exit
- =====

Enter your choice: 1

Enter number of employees: 2

===== Enter Employee 1 Details =====

Enter Employee SSN : 101

Enter Employee Name : Ravi

Enter Department : CSE

Enter Designation : Manager

Enter Salary : 50000

Enter Phone Number : 9876543210

Employee inserted at the END successfully.

===== Enter Employee 2 Details =====

Enter Employee SSN : 102

Enter Employee Name : Priya

Enter Department : ECE

Enter Designation : Engineer

Enter Salary : 45000

Enter Phone Number : 9876543211

Employee inserted at the END successfully.

DLL with 2 employees created successfully.

## =====

## DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

- =====
1. Create DLL of N Employees using End Insertion
  2. Display DLL and Count Number of Nodes
  3. Insert Employee at End
  4. Delete Employee from End
  5. Insert Employee at Front
  6. Delete Employee from Front
  7. Demonstrate DLL as Double Ended Queue
  8. Exit
- =====

Enter your choice: 2

=====

EMPLOYEE DLL

=====

| SSN | Name  | Department | Designation | Salary   | Phone No.  |
|-----|-------|------------|-------------|----------|------------|
| 101 | Ravi  | CSE        | Manager     | 50000.00 | 9876543210 |
| 102 | Priya | ECE        | Engineer    | 45000.00 | 9876543211 |

=====

Number of nodes = 2

=====

=====

DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

=====

1. Create DLL of N Employees using End Insertion
  2. Display DLL and Count Number of Nodes
  3. Insert Employee at End
  4. Delete Employee from End
  5. Insert Employee at Front
  6. Delete Employee from Front
  7. Demonstrate DLL as Double Ended Queue
  8. Exit
- =====

Enter your choice: 3

Enter Employee SSN : 103  
Enter Employee Name : Arun  
Enter Department : ISE  
Enter Designation : Developer  
Enter Salary : 40000  
Enter Phone Number : 9876543212

Employee inserted at the END successfully.

=====

DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

=====

1. Create DLL of N Employees using End Insertion
  2. Display DLL and Count Number of Nodes
  3. Insert Employee at End
  4. Delete Employee from End
  5. Insert Employee at Front
  6. Delete Employee from Front
  7. Demonstrate DLL as Double Ended Queue
  8. Exit
- =====

Enter your choice: 2

=====

## EMPLOYEE DLL

```
=====
SSN Name Department Designation Salary Phone No.

101 Ravi CSE Manager 50000.00 9876543210
102 Priya ECE Engineer 45000.00 9876543211
103 Arun ISE Developer 40000.00 9876543212
=====
```

Number of nodes = 3

## DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

- ```
=====
1. Create DLL of N Employees using End Insertion
2. Display DLL and Count Number of Nodes
3. Insert Employee at End
4. Delete Employee from End
5. Insert Employee at Front
6. Delete Employee from Front
7. Demonstrate DLL as Double Ended Queue
8. Exit
=====
```

Enter your choice: 3

```
Enter Employee SSN      : 100
Enter Employee Name     : Kiran
Enter Department        : ME
Enter Designation       : Analyst
Enter Salary            : 35000
Enter Phone Number      : 9876543213
```

Employee inserted at the END successfully.

DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

- ```
=====
1. Create DLL of N Employees using End Insertion
2. Display DLL and Count Number of Nodes
3. Insert Employee at End
4. Delete Employee from End
5. Insert Employee at Front
6. Delete Employee from Front
7. Demonstrate DLL as Double Ended Queue
8. Exit
=====
```

Enter your choice: 6

Employee deleted from FRONT:  
SSN : 101

Name : Ravi

DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

1. Create DLL of N Employees using End Insertion
2. Display DLL and Count Number of Nodes
3. Insert Employee at End
4. Delete Employee from End
5. Insert Employee at Front
6. Delete Employee from Front
7. Demonstrate DLL as Double Ended Queue
8. Exit

Enter your choice: 4

Employee deleted from END:

SSN : 100

Name : Kiran

DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

1. Create DLL of N Employees using End Insertion
2. Display DLL and Count Number of Nodes
3. Insert Employee at End
4. Delete Employee from End
5. Insert Employee at Front
6. Delete Employee from Front
7. Demonstrate DLL as Double Ended Queue
8. Exit

Enter your choice: 7

DOUBLY LINKED LIST AS DEQUE

1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display Deque
6. Return to Main Menu

Enter your choice: 1

Enter Employee SSN : 104

Enter Employee Name : Suresh

Enter Department : CSE

Enter Designation : Tester

Enter Salary : 38000

Enter Phone Number : 9876543214

Employee inserted at the FRONT successfully.

=====

DOUBLY LINKED LIST AS DEQUE

=====

1. Insert at Front
  2. Insert at Rear
  3. Delete from Front
  4. Delete from Rear
  5. Display Deque
  6. Return to Main Menu
- =====

Enter your choice: 2

Enter Employee SSN : Meena

Enter Employee Name : Meena

Enter Department : EEE

Enter Designation : Designer

Enter Salary : 42000

Enter Phone Number : 9876543215

Employee inserted at the END successfully.

=====

DOUBLY LINKED LIST AS DEQUE

=====

1. Insert at Front
  2. Insert at Rear
  3. Delete from Front
  4. Delete from Rear
  5. Display Deque
  6. Return to Main Menu
- =====

Enter your choice: 5

=====

EMPLOYEE DLL

=====

| SSN | Name   | Department | Designation | Salary   | Phone No.  |
|-----|--------|------------|-------------|----------|------------|
| 104 | Suresh | CSE        | Tester      | 38000.00 | 9876543214 |
| 102 | Priya  | ECE        | Engineer    | 45000.00 | 9876543211 |
| 103 | Arun   | ISE        | Developer   | 40000.00 | 9876543212 |
| 104 | Meena  | EEE        | Designer    | 42000.00 | 9876543215 |

=====

Number of nodes = 4

=====

DOUBLY LINKED LIST AS DEQUE

=====

1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display Deque
6. Return to Main Menu

=====

Enter your choice: 4  
Employee deleted from END:  
SSN: 104  
Name : Meena

=====

#### DOUBLY LINKED LIST AS DEQUE

=====

1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display Deque
6. Return to Main Menu

=====

Enter your choice: 3  
Employee deleted from FRONT:  
SSN : 104  
Name: Suresh

=====

#### DOUBLY LINKED LIST AS DEQUE

=====

1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display Deque
6. Return to Main Menu

=====

Enter your choice: 6

#### DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

=====

1. Create DLL of N Employees using End Insertion
2. Display DLL and Count Number of Nodes
3. Insert Employee at End
4. Delete Employee from End
5. Insert Employee at Front
6. Delete Employee from Front
7. Demonstrate DLL as Double Ended Queue

8. Exit

Enter your choice: 6

Employee deleted from FRONT:

SSN : 102

Name : Priya

### DOUBLY LINKED LIST - EMPLOYEE MANAGEMENT

1. Create DLL of N Employees using End Insertion
2. Display DLL and Count Number of Nodes
3. Insert Employee at End
4. Delete Employee from End
5. Insert Employee at Front
6. Delete Employee from Front
7. Demonstrate DLL as Double Ended Queue
8. Exit

Enter your choice: 8

Program terminated successfully.

Process returned 0 (0x0) execution time: 541.475 s

Press any key to continue.

**5. Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers.****a. Create a BST of N Integers****b. Traverse the BST in Inorder, Preorder and Post Order****c. Search the BST for a given element (KEY) and report the appropriate message****d. Exit**

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
 int data;
 struct Node *left;
 struct Node *right;
};
struct Node *createNode(int value)
{
 struct Node *newNode;
 newNode = (struct Node *)malloc(sizeof(struct Node));
 if (newNode == NULL)
 {
 printf("Memory allocation failed!\n");
 exit(1);
 }
 newNode->data = value;
 newNode->left = NULL;
 newNode->right = NULL;
 return newNode;
}
struct Node *insert(struct Node *root, int value)
{
 if (root == NULL)
 {
 return createNode(value);
 }
 if (value < root->data)
 {
 root->left = insert(root->left, value);
 }
 else if (value > root->data)
 {
 root->right = insert(root->right, value);
 }
 else
 {
 printf("Duplicate element %d is not inserted.\n", value);
 }
 return root;
}
```

```
void inorder(struct Node *root)
{
 if (root != NULL)
 {
 inorder(root->left);
 printf("%d ", root->data);
 inorder(root->right);
 }
}

void preorder(struct Node *root)
{
 if (root != NULL)
 {
 printf("%d ", root->data);
 preorder(root->left);
 preorder(root->right);
 }
}

void postorder(struct Node *root)
{
 if (root != NULL)
 {
 postorder(root->left);
 postorder(root->right);
 printf("%d ", root->data);
 }
}

struct Node *search(struct Node *root, int key)
{
 if (root == NULL || root->data == key)
 {
 return root;
 }
 if (key < root->data)
 {
 return search(root->left, key);
 }
 else
 {
 return search(root->right, key);
 }
}

void freeTree(struct Node *root)
{
 if (root != NULL)
 {
 freeTree(root->left);
 freeTree(root->right);
 }
}
```

```
 free(root);
 }
}
int main()
{
 struct Node *root = NULL;
 struct Node *result;
 char choice;
 int n;
 int value;
 int key;
 int i;
 while (1)
 {
 printf("\n=====\\n");
 printf(" BINARY SEARCH TREE MENU\\n");
 printf("=====\\n");
 printf("a. Create a BST of N Integers\\n");
 printf("b. Traverse the BST\\n");
 printf("c. Search the BST for a given KEY\\n");
 printf("d. Exit\\n");
 printf("=====\\n");
 printf("Enter your choice: ");
 scanf(" %c", &choice);
 switch (choice)
 {
 case 'a':
 case 'A':
 freeTree(root);
 root = NULL;

 printf("\\nEnter the number of integers: ");
 scanf("%d", &n);
 if (n <= 0)
 {
 printf("Number of integers must be greater than 0.\\n");
 break;
 }
 printf("Enter %d integers:\\n", n);
 for (i = 0; i < n; i++)
 {
 scanf("%d", &value);
 root = insert(root, value);
 }
 printf("\\nBST created successfully.\\n");
 break;
 case 'b':
 case 'B':
```

```
 if (root == NULL)
 {
 printf("\nBST is empty. Please create the BST first.\n");
 break;
 }
 printf("\nBST Traversals\n");
 printf("-----\n");
 printf("Inorder Traversal : ");
 inorder(root);
 printf("\n");
 printf("Preorder Traversal : ");
 preorder(root);
 printf("\n");
 printf("Postorder Traversal : ");
 postorder(root);
 printf("\n");
 break;
 case 'c':
 case 'C':
 if (root == NULL)
 {
 printf("\nBST is empty. Please create the BST first.\n");
 break;
 }
 printf("\nEnter the KEY to search: ");
 scanf("%d", &key);
 result = search(root, key);
 if (result != NULL)
 {
 printf("\nKEY %d is found in the BST.\n", key);
 }
 else
 {
 printf("\nKEY %d is not found in the BST.\n", key);
 }
 break;
 case 'd':
 case 'D':
 freeTree(root);
 root = NULL;
 printf("\nProgram terminated successfully.\n");
 return 0;
 default:
 printf("\nInvalid choice! Please enter a, b, c, or d.\n");
 }
}
return 0;
}
```

**Output:**

```
=====
 BINARY SEARCH TREE MENU
=====
a. Create a BST of N Integers
b. Traverse the BST
c. Search the BST for a given KEY
d. Exit
=====
Enter your choice: a

Enter the number of integers: 7
Enter 7 integers:
50
30
70
20
40
60
80
BST created successfully.
=====
 BINARY SEARCH TREE MENU
=====
a. Create a BST of N Integers
b. Traverse the BST
c. Search the BST for a given KEY
d. Exit
=====
Enter your choice: b
BST Traversals

Inorder Traversal : 20 30 40 50 60 70 80
Preorder Traversal : 50 30 20 40 70 60 80
Postorder Traversal : 20 40 30 60 80 70 50
=====
 BINARY SEARCH TREE MENU
=====
a. Create a BST of N Integers
b. Traverse the BST
c. Search the BST for a given KEY
d. Exit
=====
Enter your choice: c
Enter the KEY to search: 60

KEY 60 is found in the BST.
=====
```

## BINARY SEARCH TREE MENU

- ```
=====
```
- a. Create a BST of N Integers
 - b. Traverse the BST
 - c. Search the BST for a given KEY
 - d. Exit
- ```
=====
```

Enter your choice: c

Enter the KEY to search: 100

KEY 100 is not found in the BST.

```
=====
```

## BINARY SEARCH TREE MENU

- ```
=====
```
- a. Create a BST of N Integers
 - b. Traverse the BST
 - c. Search the BST for a given KEY
 - d. Exit
- ```
=====
```

Enter your choice: d

Program terminated successfully.

Process returned 0 (0x0) execution time : 117.953 s

Press any key to continue.

**6. Develop a Program in C for an Employee Record Management System using Hashing with the following operations:**

- a. Create a hash table to store employee records using Employee ID as the key.
- b. Insert employee records into the hash table using the Hash function  $H: K \rightarrow L$  as  $H(K) = K \bmod m$
- c. Resolve collisions using Linear Probing.
- d. Search for an employee record using a given Employee ID and display the details.

**Display the contents of the hash table along with the corresponding memory locations.**

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
struct Employee
{
 int empID;
 char name[50];
 float salary;
 int occupied;
};
int hashFunction(int key)
{
 return key % SIZE;
}
void insert(struct Employee hashTable[], int empID, char name[], float salary)
{
 int index = hashFunction(empID);
 int originalIndex = index;

 while (hashTable[index].occupied)
 {
 index = (index + 1) % SIZE;
 if (index == originalIndex)
 {
 printf("Hash table is full. Cannot insert Employee ID %d.\n", empID);
 return;
 }
 }
 hashTable[index].empID = empID;
 snprintf(hashTable[index].name,
 sizeof(hashTable[index].name), "%s", name);
 hashTable[index].salary = salary;
 hashTable[index].occupied = 1;
 printf("Employee ID %d inserted at index %d.\n", empID, index);
}
void search(struct Employee hashTable[], int empID)
{
 int index = hashFunction(empID);
 int originalIndex = index;
```

```
while (hashTable[index].occupied)
{
 if (hashTable[index].empID == empID)
 {
 printf("\nEmployee Record Found\n");
 printf("-----\n");
 printf("Employee ID : %d\n", hashTable[index].empID);
 printf("Name : %s\n", hashTable[index].name);
 printf("Salary : %.2f\n", hashTable[index].salary);
 printf("Hash Index : %d\n", index);
 printf("Memory Addr : %p\n", (void *)&hashTable[index]);
 return;
 }
 index = (index + 1) % SIZE;

 if (index == originalIndex)
 break;
}
printf("\nEmployee ID %d not found in the hash table.\n", empID);
}
void display(struct Employee hashTable[])
{
 int i;
 printf("\n===== HASH TABLE =====\n");
 printf("%-8s %-12s %-20s %-12s %-18s\n",
 "Index",
 "Employee ID",
 "Name",
 "Salary",
 "Memory Location");
 printf("-----\n");
 for (i = 0; i < SIZE; i++)
 {
 if (hashTable[i].occupied)
 {
 printf("%-8d %-12d %-20s %-12.2f %p\n",
 i,
 hashTable[i].empID,
 hashTable[i].name,
 hashTable[i].salary,
 (void *)&hashTable[i]);
 }
 else
 {
 printf("%-8d %-12s %-20s %-12s %p\n",
 i,
 "EMPTY",
 "___",
 "___",
 (void *)&hashTable[i]);
 }
 }
}
```

```
 "___",
 (void *)&hashTable[i]);
 }
}
printf("=====\n");
}
int main()
{
 struct Employee hashTable[SIZE];
 int i, choice, empID;
 char name[50];
 float salary;
 for (i = 0; i < SIZE; i++)
 {
 hashTable[i].occupied = 0;
 }
 printf("=====\n");
 printf(" EMPLOYEE RECORD MANAGEMENT SYSTEM\n");
 printf(" HASHING - LINEAR PROBING\n");
 printf("=====\n");
 while (1)
 {
 printf("\n1. Insert Employee Record");
 printf("\n2. Search Employee Record");
 printf("\n3. Display Hash Table");
 printf("\n4. Exit");
 printf("\n\nEnter your choice: ");
 scanf("%d", &choice);
 switch (choice)
 {
 case 1:
 printf("\nEnter Employee ID: ");
 scanf("%d", &empID);
 printf("Enter Employee Name: ");
 scanf(" %49[^\n]", name);
 printf("Enter Salary: ");
 scanf("%f", &salary);
 insert(hashTable, empID, name, salary);
 break;
 case 2:
 printf("\nEnter Employee ID to search: ");
 scanf("%d", &empID);
 search(hashTable, empID);
 break;
 case 3:
 display(hashTable);
 break;
 case 4:
```

```
 printf("\nProgram terminated successfully.\n");
 return 0;
 default:
 printf("\nInvalid choice! Please try again.\n");
 }
}
return 0;
}
```

**Output:**

## =====

EMPLOYEE RECORD MANAGEMENT SYSTEM  
HASHING - LINEAR PROBING

=====

1. Insert Employee Record
2. Search Employee Record
3. Display Hash Table
4. Exit

Enter your choice: 1  
Enter Employee ID: 101  
Enter Employee Name: Ravi  
Enter Salary: 35000  
Employee ID 101 inserted at index 1.

1. Insert Employee Record
2. Search Employee Record
3. Display Hash Table
4. Exit

Enter your choice: 1  
Enter Employee ID: 111  
Enter Employee Name: priya  
Enter Salary: 42000  
Employee ID 111 inserted at index 2.

1. Insert Employee Record
2. Search Employee Record
3. Display Hash Table
4. Exit

Enter your choice: 1  
Enter Employee ID: 121  
Enter Employee Name: arun  
Enter Salary: 38000  
Employee ID 121 inserted at index 3.

1. Insert Employee Record
2. Search Employee Record
3. Display Hash Table
4. Exit

Enter your choice: 1  
Enter Employee ID: 102  
Enter Employee Name: kiran  
Enter Salary: 45000  
Employee ID 102 inserted at index 4.

1. Insert Employee Record
2. Search Employee Record
3. Display Hash Table
4. Exit

Enter your choice: 3

===== HASH TABLE =====

| Index | Employee ID | Name  | Salary   | Memory Location |
|-------|-------------|-------|----------|-----------------|
| 0     | EMPTY       | ---   | ---      | 0061FC7C        |
| 1     | 101         | Ravi  | 35000.00 | 0061FCBC        |
| 2     | 111         | priya | 42000.00 | 0061FCFC        |
| 3     | 121         | arun  | 38000.00 | 0061FD3C        |
| 4     | 102         | kiran | 45000.00 | 0061FD7C        |
| 5     | EMPTY       | ---   | ---      | 0061FDBC        |
| 6     | EMPTY       | ---   | ---      | 0061FDFC        |
| 7     | EMPTY       | ---   | ---      | 0061FE3C        |
| 8     | EMPTY       | ---   | ---      | 0061FE7C        |
| 9     | EMPTY       | ---   | ---      | 0061FEBC        |

1. Insert Employee Record  
2. Search Employee Record  
3. Display Hash Table  
4. Exit

Enter your choice: 2

Enter Employee ID to search: 121  
Employee Record Found

-----  
Employee ID : 121  
Name : arun  
Salary : 38000.00  
Hash Index : 3  
Memory Addr : 0061FD3C

1. Insert Employee Record  
2. Search Employee Record  
3. Display Hash Table  
4. Exit

Enter your choice: 2

Enter Employee ID to search: 999  
Employee ID 999 not found in the hash table.

1. Insert Employee Record  
2. Search Employee Record  
3. Display Hash Table  
4. Exit

Enter your choice: 4  
Program terminated successfully.  
Process returned 0 (0x0) execution time : 161.340 s  
Press any key to continue.

---

## VIVA QUESTIONS WITH ANSWERS

### 1. What is Data Structure?

A data structure is a model for organizing and storing data. Stacks, Queue, Linked Lists, and Trees are the examples of different data structures. Data structures are classified as either linear or non-linear:

A data structure is said to be linear if only one element can be accessed from an element directly. Eg: Stacks, Lists, Queues.

A data structure is non-linear if more than one elements can be accessed from an element directly. Eg: Trees, Graphs, Heap.

### 2. Define ADT?

An abstract data type is a set of objects together with a set of operations. Abstract data types are mathematical abstraction. Objects such as lists, sets, graphs, trees can be viewed as abstract data types.

### 3. What do you mean by LIFO and FIFO?

LIFO stands for 'Last In First Out', it says how the data to be stored and retrieved. It means the data or element which was stored last should come out first. Stack follows LIFO.

FIFO stands for 'First In First Out', here the data(or element) which was stored first should be the one to come out first. It is implemented in Queue.

### 4. What is Stack?

A stack is a list of elements in which insertion and deletions can take place only at one end, this end is called as stack 'top'. Only top element can be accessed. A stack data structure has LIFO (Last In First Out) property. A stack is an example of linear data structure.

### 5. What operations can be done on Stack?

The following operations can be performed on stack:

Push operation inserts new element into stack.

Pop operation deletes the top element from the stack.

Peek returns or give the top element without deleting it.

Is empty() operation returns whether stack is empty or not.

### 6. List some applications of stack?

Some well-known applications of stack are as follow:

Infix to Postfix conversion.

Evaluation of postfix expression.

Check for balanced parantheses in an expression.

Stack is very important data structure being used to implement function calls efficiently.

Parsing

Simulation of recursion function.

String reverse using stack.

### 7. What is a Queue?

A Queue is a particular data structure in which the elements in the collection are kept in order. Unlike Stack, queue is opened at both end. Queue follows 'FIFO' method where element is inserted from rear end and deleted or removed from front end.

### 8. What operations can be done on Queue?

The following operations can be performed on queue:

Enqueue operation inserts new element in rear of the list.  
Dequeue operation deletes the front element from the list.  
IsEmpty() operation checks whether queue is empty or not.

**9. Where do we use Queue?**

Some well-known applications of queue are as follow:  
For finding level order traversal efficiently.  
For implementing BFS efficiently.  
For implementing Kruskal algorithm efficiently.

**10. What is Binary Tree?**

A binary tree is a 2-ary tree in which each node(N) has at most 2 children (either 0 or 1). The node with 2 children are called internal nodes, and the nodes with 0 children are called external nodes or leaf nodes.

**11. What is Complete Binary Tree?**

A Binary tree is complete binary tree if all levels are completely filled except possibly the last/lowest level are full and in the last/lowest level all the items are on the left.

**12. What is Perfect Balanced Binary Tree?**

A perfect balanced binary tree is a binary tree where each node has same number of nodes in both subtrees.

**13. Define Height in a tree?**

Height is a general measured as number of edges from the root to deepest node in the tree.

**14. What is tree traversal?**

Traversal is used to visit each node in the tree exactly once. A full traversal of a binary tree gives a linear ordering of the data in the tree. There are 3 different type of tree traversal:

Inorder Traversal  
Preorder Traversal  
Postorder Traversal

**15. How does Inorder Traversal work?**

Traverse the left sub tree of the root node R in inorder.  
Visit the root node R.  
Traverse the right sub tree of the root node R in inorder.

**16. How does Preorder Traversal work?**

Traverse the left sub tree of the root node R in preorder.  
Traverse the right sub tree of the root node R in preorder.  
Visit the root node R.

**17. How does Postorder Traversal work?**

Visit the root node R.  
Traverse the left sub tree of the root node R in postorder.  
Traverse the right sub tree of the root node R in postorder.

**18. What is a Binary Search Tree?**

A BST(Binary Search Tree) is a binary tree in which each node satisfies search property. Each node's key value must be greater than all values in its left subtree and must be less than all values in its right subtree.

**19. What is a AVL Tree?**

AVL tree is a self-balancing binary search tree with height-balanced condition. For every node the height of left subtree and right subtree can be differ by at most 1.

**20. What is a B-tree?**

A B-Tree is a tree data structure that keeps data sorted and allows searches, insertions, deletions, and sequential access in logarithmic amount of time.

In B-Tree, internal nodes can have a variable number of child nodes within some pre-defined range.

A B-tree is kept balanced by requiring that all leaf nodes are at the same depth.

A B-tree of order  $m$  is a tree which satisfies the following properties:

Every node has almost  $2m$  children.

Every node (except root and leaves) has at least  $m$  children.

The root has at least two children if it is not a leaf node.

All leaves appear in the same level, and carry information.

A non-leaf node with  $k$  children contain  $k-1$  keys.

Leaf nodes contain at least  $m-1$  children and almost  $2m-1$  keys.

**21. What is a B+ Tree?**

A B+ Tree is a tree data structure which represents sorted data in a way that allows for efficient insertion, retrieval and removal of records, each of which is identified by a key. It is a dynamic multilevel index, with maximum and minimum bounds on the number of keys in each segment (usually called a 'block' or 'node')

**22. What is a Binary Heap Tree?**

A Binary Heap tree is a balanced binary tree where root node are compared with its children and placed accordingly. Heap have two properties namely, structure and order property.

Structure Property

It is a complete binary tree where all the levels except possibly the last/lower level are full and in the last/lower level all the items are on the left. Due to this fact that binary heap is a complete binary it can be implemented using a simple array.

Order Property

The heap order property for MinHeap is 'a parent is less than (or equal to) its children' whereas for MaxHeap 'a parent is greater than its children'.

**23. What is a bubble sort?**

Bubble sort is a simple sorting algorithm, it works by repeatedly stepping through the list to be sorted comparing each pair of adjacent items and swapping if they are in the wrong order.

First pass bubbles out the largest element and places it in the last position and second pass places the second largest element in the second last position and so on. Thus in the last pass smallest element is placed in the first position.

The running time is the total number of comparisons that is  $n(n-1)/2$  which implies  $O(n^2)$  time complexity.

**24. What is Insertion Sort?**

Insertion sort is a simple comparison sorting algorithm, every iteration of insertion sort removes an element from the input data and insert it into the correct position in the already sorted list until no input element remains.

The running time is the total number of comparisons that is  $n(n-1)/2$  which implies  $O(n^2)$  time complexity.

### 25. What is Selection Sort?

Insertion sort is a simple comparison sorting algorithm, the algorithm works as follows:

Find the minimum value in the list.

Swap it with the minimum value in the first position.

Repeat the steps above for the remainder of the list (starting at the second position and advancing each time).

The running time is the total number of comparisons that is  $n(n-1)/2$  which implies  $O(n^2)$  time complexity.

### 26. What is Merge Sort?

Merge sort is an  $O(n \log n)$  comparison-based divide and conquer sorting algorithm, it works as follows:

If the list is of length 0 or 1, then it is already sorted.

Divide the unsorted list into two sub lists of about half the size.

Sort each sub list recursively by re-applying merge sort algorithm.

Merge the two sub lists back into one sorted list.

If the running time of merge sort for a list of length  $n$  is  $T(n)$  then the recurrence relation is  $T(n) = 2T(n/2) + n$ , thus after simplifying the recurrence relation  $T(n) = O(n \log n)$ .

### 27. What is Heap Sort?

Heap Sort is a comparison-based sorting algorithm which is much more efficient version of selection sort, it works by determining the largest (or smallest) element of the list, placing that at the end (or beginning) of the list, then continuing with the rest of the list, but accomplishes this task efficiently by using a data structure called a heap. Once the data list has been made into a heap, the root node is guaranteed to be the largest (or smallest) element.

### 28. What is Quick Sort?

Quick sort sorts by employing a divide and conquer strategy to divide a list into two sub-lists. The steps are:

Pick an element, called a pivot, from the list.

Reorder the list so the elements which are less than the pivot come before the pivot and the elements greater than pivot come after it. After this partitioning the pivot is in its final position.

Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

The running time complexity for worst case is  $O(n^2)$  and for best and average case it is same i.e.,  $O(n \log n)$ .

### 29. What is a Graph?

A graph is a pair of sets  $(V, E)$ , where  $V$  is the sets of vertices and  $E$  is the set of edges connecting the pair of vertices.

### 30. What is a Directed and Undirected Graph?

A graph is directed if each edge of graph has a direction between vertices.

A graph is undirected if there are no direction between vertices.

**31. What is a Connected Graph?**

A undirected graph is connected if there is a path from every vertex to every other vertex. A directed graph with this property is called strongly connected.

**32. What is a Complete Graph?**

A complete graph is a graph in which every vertex is connected to every other vertex. That means each vertex's degree in undirected graph is  $n-1$ .

**33. What is a Acyclic Graph?**

An acyclic graph is a graph which does not have cycles.

**34. How are graph represented?**

A graph can be represented in two forms 'Adjacency matrix' and 'Adjacency list'.

Adjacency matrix is a two dimensional arrays where for each edge,  $(u,v)$   $A[u,v] = \text{true}$  otherwise it will be false. The space complexity to represent a graph using this is  $O(|V|^2)$ .

Adjacency list are used where each list stores adjacent vertices of the corresponding vertex. The space complexity to represent a graph using this is  $O(|V|+|E|)$ .

**35. What is a Spanning Tree?**

A Spanning tree is a graph which must include every vertex and if the graph contain  $n$  vertices, spanning tree must contain exactly  $(n-1)$  edges and is a subgraph of  $G$ .

**36. What is a Minimum Spanning Tree?**

A Minimum Spanning tree is a spanning tree such that the sum of all the weights of edges in spanning tree is minimum.

**37. Explain algorithm for finding Minimum Spanning Tree?**

There are 2 widely used algorithm for finding minimum spanning tree:

Prim's Algorithm

Kruskal Algorithm

Prim's Algorithm computes the minimum spanning tree by including appropriate vertex and thus one edge into existing partially constructed tree in successive stages. The time complexity of the Prim's Algorithm is  $O((V+E)\log V)$ .

Kruskal Algorithm computes the minimum spanning tree by adding edges one by one into a growing spanning tree.

Initially from the graph  $G$ , consider a forest of all the vertices without any edge.

Sort all the edges in ascending order to their costs.

Include the edge with minimum cost into the forest if it does not form a cycle in the partially constructed tree.

Repeat step (2) until no edge can be added to the tree.

**38. List one application of Hashing.**

Hashing is used to construct the symbol table used by the language compilers.