

ATME COLLEGE OF ENGINEERING

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



A T M E
College of Engineering

DEPARTMENT OF CSE-CYBER SECURITY

(ACADEMIC YEAR 2026-27)

LABORATORY MANUAL

SUBJECT: DATA STRUCTURES LABORATORY

SUB CODE: 1BCSL306

SEMESTER: III-2025 CBCS Scheme

Composed by

Verified by

Approved by

Mr. Mahadevaswamy D M

Ms. Bhavana V R

Dr. NASREEN FATHIMA

PROGRAMMER

FACULTY CO-ORDINATOR

HOD, CSE-CY

INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, Technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

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

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 MISSION AND VISION

Vision

To nurture competent and ethical cyber security professionals capable of contributing to secure and resilient digital ecosystems.

Mission

- M1: To impart strong foundational knowledge in cyber security through effective teaching–learning practices and a continuously evolving curriculum that addresses emerging cyber threats and technologies.
- M2: To promote research, innovation, and industry collaboration in emerging areas of cyber security to address real-world digital security challenges.
- M3: To instill professional ethics, teamwork, leadership qualities, and a sense of social responsibility in addressing real-world cyber challenges.
- M4: To foster a culture of lifelong learning among students by encouraging higher studies, professional development, and active engagement with the global cyber security community

Program Specific Outcome

PSO1 – Apply principles of computer science, programming, algorithms, operating systems, and computer networks to design and develop efficient and reliable computing solutions.

PSO2 – Identify, analyze, and mitigate cyber threats and vulnerabilities using techniques related to network security, ethical hacking, penetration testing, and digital forensics.

PSO3 – Apply cyber security policies, cyber laws, risk management practices, and incident response strategies to ensure the protection and resilience of digital systems.

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

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.

Note:

1. The laboratory syllabus consists of PART-A and PART-B. While PART-A has 6 conventional experiments, PART-B has 6 typical open-ended experiments. The maximum marks for the laboratory course are 100.
- 2 Both PART-A and PART-B are considered for CIE and SEE.
3. Students have answer 1(one) question from PART-A and 1(one) question from PART-B.
 - a. The questions set for SEE shall be from among the experiments under PART-A. It is evaluated for 70 marks out of the maximum 100 marks.
 - b. The open-ended question set for SEE shall be any other open-ended question and not selected from the experiments under PART-A. It shall be evaluated for 30 marks.
4. For Continuous Internal Evaluation (CIE) during the semester, classwork shall typically include open-ended questions from Part-B, along with other similar questions designed to enhance the students' skills.

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

Reference books / Manuals:

- 1. Reema Thareja, Data Structures using C, 3rd Ed, Oxford press, 2023.
- 2. A M Tenenbaum, Data Structures using C, PHI, 1989
- 3. Seymour Lipschutz, Data Structures Schaum's Outlines, Revised 1st Ed, McGraw Hill, 2014.

Teaching-Learning Process (Innovative Delivery Methods):

The following are sample strategies that educators may adopt to enhance the effectiveness of the teaching-learning process and facilitate the achievement of course outcomes.

- 1. Engineering tool usage for the conduction of experiment
- 2. Demonstration through ICT tools
- 3. Use of virtual labs

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.

CONTENTS

Sl.No.	EXPERIMENT NAME	Page No
	Introduction	1-5
	PART-A	
1.	Program 1 : Develop a program in Define a structure Book with fields:	5-12
2.	Program 2 : Develop a menu driven Program in C for the following operations on STACK of Integers	13-22
3.	Program 3 : Develop a program in C for simulating a Printer Queue using Queue data structure with the operations:	23-28
4.	Program 4: Singly linked list	29-36
5.	Program 5: Binary tree traversal	37-43
6.	Program 6 :: graphs DFS/BFS Method	44-49
	PART B:	
1	Program 1: Develop a C program to sparse matrices, Resultant matrices	50-54
2.	Program 2: Conversions infix to postfix	55-60
3	program 3 : Circular queue	61-66
4	Program 4 : doubly linked list	67-64
5	Program 5 : binary search tree	75-80
6	Program 6 : Employee Record Management System using Hashing with the operations:	80-86

Introduction to Data Structure

Basic Concepts

The logical or mathematical model of a particular organization of data is called data structures. Data structures is the study of logical relationship existing between individual data elements, the way the data is organized in the memory and the efficient way of storing, accessing and manipulating the data elements.

Data Structures can be classified as:

- Primitive data structures
- Non-Primitive data structures.

Primitive data structures are the basic data structures that can be directly manipulated/operated by machine instructions. Some of these are character, integer, real, pointers etc.

Non-primitive data structures are derived from primitive data structures, they cannot be directly manipulated/operated by machine instructions, and these are group of homogeneous or heterogeneous data items. Some of these are Arrays, stacks, queues, trees, graphs etc.

Data structures are also classified as

- Linear data structures
- Non-Linear data structures.

In the Linear data structures processing of data items is possible in linear fashion, i.e., data can be processed one by one sequentially.

Example of such data structures are:

- Array
- Linked list
- Stacks
- Queues

A data structure in which insertion and deletion is not possible in a linear fashion is called as non linear data structure. i.e., which does not show the relationship of logical adjacency between the elements is called as non-linear data

structure. Such as trees, graphs and files.

Data structure operations:

The particular data structures that one chooses for a given situation depends largely on the frequency with which specific operations are performed.

The following operations play major role in the processing of data.

- i) Traversing.
- ii) Searching.
- iii) Inserting.
- iv) Deleting.
- v) Sorting.
- vi) Merging

STACKS:

A stack is an ordered collection of items into which new items may be inserted and from which items may be deleted at the same end, called the TOP of the stack. A stack is a non-primitive linear data structure. 1 2 3 4 5

As all the insertion and deletion are done from the same end, the first element inserted into the stack is the last element deleted from the stack and the last element inserted into the stack is the first element to be deleted. Therefore, the stack is called Last-In First-Out (LIFO) data structure.

QUEUES:

A queue is a non-primitive linear data structure. Where the operation on the queue is based on First-In-First-Out FIFO process — the first element in the queue will be the first one out. This is equivalent to the requirement that whenever an element is added, all elements that were added before have to be removed before the new element can be removed.

For inserting elements into the queue are done from the rear end and deletion is done from the front end, we use external pointers called as rear and front to keep track of the status of the queue. During insertion, Queue Overflow condition has to be checked. Likewise during deletion, Queue Underflow condition is checked.

APPLICATION OF QUEUE

Queue, as the name suggests is used whenever we need to have any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios :

- Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
- In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.
- Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, First come first served.

LINKED LIST**Disadvantages of static/sequential allocation technique:**

- If an item has to be deleted then all the following items will have to be moved by one allocation. Wastage of time.
- Inefficient memory utilization.
- If no consecutive memory (free) is available, execution is not possible.

Linear Linked Lists

Types of Linked lists:

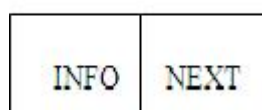
- Single Linked lists
- Circular Single Linked Lists
- Double Linked Lists
- Circular Double Linked Lists.

NODE:

Each node consists of two fields. Information (info) field and next address (next) field. The info field consists of actual information/data/item that has to be stored in a list. The second field next/link contains the address of the next node. Since next field contains the address,

It is of type pointer. Here the nodes in the list are logically adjacent to each other. Nodes that are physically adjacent need not be logically adjacent in the list.

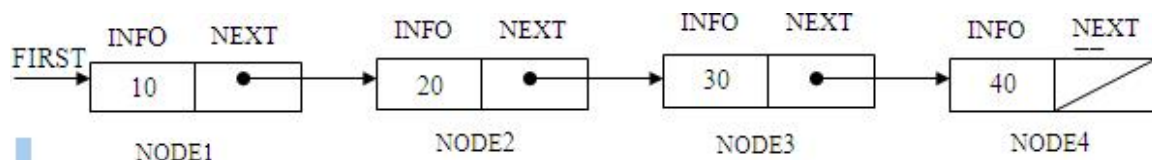
The entire linked list is accessed from an external pointer FIRST that points to (contains the address of) the first node in the list. (By an —external pointer, we mean, one that is not included

NODE

within a node. Rather its value can be accessed directly by referencing a variable).

Fig-1 Linked List

The list containing 4 items/data 10, 20, 30 and 40 is shown below.

**Fig-2 Linked List**

The nodes in the list can be accessed using a pointer variable. In the above fig. FIRST is the pointer having the address of the first node of the list, initially before creating the list, as list is

empty. The FIRST will always be initialized to NULL in the beginning. Once the list is created, FIRST contains the address of the first node of the list.

As each node is having only one link/next, the list is called single linked list and all the nodes are linked in one direction. Each node can be accessed by the pointer pointing (holding the address) to that node, Say P is pointer to a particular node, then the information field of that node can be accessed using info(P) and the next field can be accessed using next(P).

The arrows coming out of the next field in the fig. indicates that the address of the succeeding node is stored in that field. The link field of last node contains a special value known as NULL which is shown using a diagonal line pictorially. This NULL pointer is used to signal the end of a list.

The basic operations of linked lists are Insertion, Deletion and Display. A list is a dynamic data structure. The number of nodes on a list may vary dramatically as elements are inserted and deleted(removed).

The dynamic nature of list may be contrasted with the static nature of an array, whose size remains constant. When an item has to be inserted, we will have to create a node, which has to be got from the available free memory of the computer system, So we shall use a mechanism to find an unused node which makes it available to us. For this purpose we shall use the getnode operation (getnode() function).

The C language provides the built-in functions like malloc(), calloc(), realloc() and free(), which are stored in alloc.h or stdlib.h header files. To dynamically allocate and release the memory locations from/to the computer system.

TREES:

Definition:

A data structure which is accessed beginning at the root node. Each node is either a leaf or an internal node. An internal node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings. Contrary to a physical tree, the root is usually depicted at the top of the structure, and the leaves are depicted at the bottom. A tree can also be defined as a connected, acyclic di-graph.

Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition. A tree data structure can also be defined as follows...

Tree data structure is a collection of data (Node) which is organized in hierarchical structure and this is a recursive definition

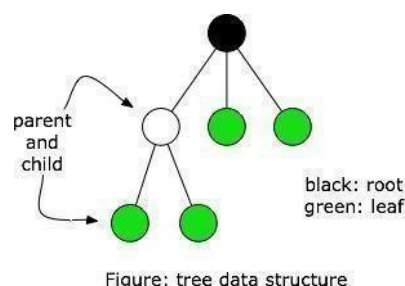


Fig-3 Tree data structure

Binary tree: A tree with utmost two children for each node.

Complete Binary Tree: A binary tree in which every level, except possibly the deepest, is completely filled. At depth n , the height of the tree, all nodes must be as far left as possible.

Binary search tree: A binary tree where every node's left subtree has keys less than the node's key, and every right subtree has keys greater than the node's key.

Tree traversal is a technique for processing the nodes of a tree in some order. The different tree traversal techniques are Pre-order, In-order and Post-order traversal. In Pre-order traversal, the tree node is visited first and the left subtree is traversed recursively and later right sub-tree is traversed recursively.

PROGRAM 1**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**

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

```
// Define the Book structure
struct Book {
int book_id;
char title[50];
char author[50];
float price;
char status[20]; // "Available" or "Issued"
};
```

```
// Global pointers and variables for dynamic memory management
struct Book *books = NULL;
int n = 0;
// Function declarations
void create();
void display();
void search();
void issueBook();
void returnBook();
int main() {
int choice;
do {
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: ");
scanf("%d", &choice);
switch (choice)
{
    case 1:
        create();
        break;
    case 2:
        display();
        break;
    case 3:
        search();
        break;
    case 4:
        issueBook();
        break;
    case 5:
        returnBook();
        break;
    case 6:
        printf("Exiting program. Freeing allocated memory...\n");
        if (books != NULL) {
            free(books);
        }
        break;
    default:
        printf("Invalid choice! Please try again.\n");
}
}
while (choice != 6);

return 0;
}

```

// i. create() – Allocate memory and read details of N books

```

void create() {
    printf("Enter the number of books (N): ");
    scanf("%d", &n);
    // Dynamically allocate memory for N books
    books = (struct Book *)malloc(n * sizeof(struct Book));
    if (books == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    for (int i = 0; i < n; i++)
    {
        printf("\nEnter details for Book %d:\n", i + 1);
    }
}

```

```

printf("Book ID: ");
scanf("%d", &books[i].book_id);
printf("Title: ");
scanf(" %[^\\n]s", books[i].title); // Read string with spaces
printf("Author: ");
scanf(" %[^\\n]s", books[i].author);
printf("Price: ");
scanf("%f", &books[i].price);
strcpy(books[i].status, "Available");
}
printf("Book records added successfully!\\n");
}
// ii. display() – Display details of all books in the library
void display() {
    if (books == NULL || n == 0) {
        printf("\\nNo book records found. Please add books first.\\n");
        return;
    }
    printf("\\n--- Library Books Details ---\\n");
    printf("%-10s %-25s %-20s %-10s %-15s\\n", "Book ID", "Title", "Author", "Price", "Status");
    printf("-----\\n");
    for (int 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].status);
    }
}
// iii. search() – Search for a book using Book ID
void search() {
    if (books == NULL || n == 0) {
        printf("\\nNo book records available to search.\\n");
        return;
    }
    int target_id, found = 0;
    printf("Enter Book ID to search: ");
    scanf("%d", &target_id);
    for (int i = 0; i < n; i++) {
        if (books[i].book_id == target_id)
        { printf("\\nBook Found!\\n");
          printf("Book ID: %d\\n", books[i].book_id);
          printf("Title: %s\\n", books[i].title);
          printf("Author: %s\\n", books[i].author);
          printf("Price: %.2f\\n", books[i].price);
          printf("Status: %s\\n", books[i].status);
          found = 1;
          break;
        }
    }
}

```

```
}
If (!found) {
    printf("Book with ID %d not found.\n", target_id);
}
}

// iv. issueBook() – Issue a book by updating availability status
void issueBook() {
    if (books == NULL || n == 0)
    { printf("\nNo book records
available.\n"); return;
    }
    int target_id, found = 0;
    printf("Enter Book ID to issue: ");
    scanf("%d", &target_id);
    for (int i = 0; i < n; i++) {
        if (books[i].book_id == target_id)
        { found = 1;
        if (strcmp(books[i].status, "Available") == 0)
        { strcpy(books[i].status, "Issued");
        printf("Success! Book '%s' has been issued.\n", books[i].title);
        }
        else
        {
        printf("Sorry, the book '%s' is already issued.\n", books[i].title);
        }
        break;
        }
    }
    if (!found)
    {
        printf("Book with ID %d not found.\n", target_id);
    }
}

// v. returnBook() – Return an issued book by updating availability status
void returnBook() {
    if (books == NULL || n == 0)
    { printf("\nNo book records available.\n");
    return;
    }
    int target_id, found = 0;
    printf("Enter Book ID to return: ");
    scanf("%d", &target_id);
    for (int i = 0; i < n; i++)
    {
        if (books[i].book_id == target_id)
        { found = 1;
        if (strcmp(books[i].status, "Issued") == 0)
        {
            strcpy(books[i].status, "Available");
            printf("Success! Book '%s' has been returned and is now available.\n", books[i].title);
        }
    }
```

```
    else
    {
printf("Error! The book '%s' was not marked as issued.\n", books[i].title);
    }
    break;
    }
    }
if (!found)
{
printf("Book with ID %d not found.\n", target_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

Enter the number of books (N): 2

Enter details for Book 1:

Book ID: 101

Title: padmareddy

Author: ds

Price: 10000

Enter details for Book 2:

Book ID: 102

Title: excellent

Author: ds

Price: 5000

Book records added 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: 2

--- Library Books Details ---

Book ID	Title	Author	Price	Status
---------	-------	--------	-------	--------

101	padmareddy	ds	10000.00	Available
102	excellent	ds	5000.00	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: 3

Enter Book ID to search: 101

Book Found!

Book ID: 101

Title: padmareddy

Author: ds

Price: 10000.00

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: 4

Enter Book ID to issue: 102

Success! Book 'excellent' has been issued.

--- 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: 2

--- Library Books Details ---

Book ID	Title	Author	Price	Status
101	padmareddy	ds	10000.00	Available
102	excellent	ds	5000.00	Issued

--- 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: 3
Enter Book ID to search: 101

Book Found!
Book ID: 101
Title: padmareddy
Author: ds
Price: 10000.00
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: 3
Enter Book ID to search: 102

Book Found!
Book ID: 102
Title: excellent
Author: ds
Price: 5000.00

Status: Issued

--- 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

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
- code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int stack[MAX];
int top = -1;

/* Function to push an element onto the stack */
void push(int item)
{
    if (top == MAX - 1)
    {
        printf("\nStack Overflow! Cannot push %d.\n", item);
    }
    else
    {
        top++;
        stack[top] = item;
        printf("\n%d pushed onto the stack.\n", item);
    }
}

/* Function to pop an element from the stack */
void pop()
{
    if (top == -1)
    {
        printf("\nStack Underflow! Stack is empty.\n");
    }
    else
    {
        printf("\n%d popped from the stack.\n", stack[top]);
        top--;
    }
}
```

```
/* Function to display stack from TOP to BOTTOM */
void display()
{
    int i;
    if (top == -1)
    {
        printf("\nStack is empty.\n");
        return;
    }
    printf("\nStack elements from TOP to BOTTOM:\n");
    printf("TOP -> ");
    for (i = top; i >= 0; i--)
    {
        printf("%d", stack[i]);
        if (i != 0)
        {
            printf(" -> ");
        }
    }
    printf(" -> BOTTOM\n");
}

/* Function to check palindrome using stack */
void palindrome()
{
    int num, temp, digit;
    int palStack[MAX];
    int palTop = -1;
    int original;
    int reversed = 0;
    printf("\nEnter an integer to check for palindrome: ");
    scanf("%d", &num);
    if (num < 0)
    {
        printf("\nNegative numbers are not considered palindromes.\n");
        return;
    }
    original = num;
    temp = num;
    /* Special case for 0 */
    if (num == 0)
    {
        printf("\n0 is a Palindrome.\n");
        return;
    }
}
```

```
    }
    /* Push each digit onto the stack */
    while (temp != 0)
    {
        digit = temp % 10;
        if (palTop == MAX - 1)
        {
            printf("\nNumber has more than %d digits.\n", MAX);
            return;
        }
        palTop++;
        palStack[palTop] = digit;
        temp = temp / 10;
    }

    /* Pop digits and form the reversed number */
    while (palTop != -1)
    {
        reversed = reversed * 10 + palStack[palTop]; palTop--;
    }
    /* Compare original and reversed number */
    if (original == reversed)
    {
        printf("\n%d is a Palindrome.\n", original);
    }
    else
    {
        printf("\n%d is NOT a Palindrome.\n", original);
    }
}

/* Function to demonstrate stack overflow */
void demonstrateOverflow()
{
    int i;
    int oldTop;
    int oldStack[MAX];
    printf("\n--- Demonstrating Stack Overflow ---\n");
    /*
    Save the current stack so that the demonstration
    does not destroy the user's stack.
    */
    oldTop = top;
```

```
for (i = 0; i <= top; i++)
{
    oldStack[i] = stack[i];
}
/* Make stack empty temporarily */
top = -1;
/* Push MAX + 1 elements */
for (i = 1; i <= MAX + 1; i++)
{
    push(i);
}
/*
Restore the original stack
*/
top = oldTop;
for (i = 0; i <= top; i++)
{
    stack[i] = oldStack[i];
}
printf("\nOriginal stack restored.\n");
}
/* Function to demonstrate stack underflow */
void demonstrateUnderflow()
{
    printf("\n--- Demonstrating Stack Underflow ---\n");
    /*
    Try to pop without changing the current stack.
    We use a temporary stack state.
    */
    {
        int oldTop = top;
        top = -1;
        pop();
        top = oldTop;
    }
    printf("\nOriginal stack preserved.\n");
}
/* Main function */
int main()
{
    int choice;
    int item;
```

```
while (1)
{
    printf("\n\n=====\\n");
    printf("    STACK MENU\\n");
    printf("=====\\n");
    printf("1. Push an Element\\n");
    printf("2. Pop an Element\\n");
    printf("3. Check Palindrome using Stack\\n");
    printf("4. Demonstrate Overflow and Underflow\\n");
    printf("5. Display Stack Status\\n");
    printf("6. Exit\\n");
    printf("=====\\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1:
            printf("\nEnter the element to push: ");
            scanf("%d", &item);
            push(item);
            break;
        case 2:
            pop();
            break;
        case 3:
            palindrome();
            break;
        case 4:
            demonstrateOverflow();
            demonstrateUnderflow();
            break;
        case 5:
            display();
            break;
        case 6:
            printf("\nExiting program...\\n");
            exit(0);
        default:
            printf("\nInvalid choice! Please enter 1 to 6.\\n");
    }
}
return 0;
}
```

*****OUTPUT*****

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 1

Enter the element to push: 10

10 pushed onto the stack.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 1

Enter the element to push: 20

20 pushed onto the stack.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice:

1

Enter the element to push: 30

30 pushed onto the stack.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 1

Enter the element to push: 40

40 pushed onto the stack.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 1

Enter the element to push: 50

50 pushed onto the stack.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 2

50 popped from the stack.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 2

40 popped from the stack.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 2

30 popped from the stack.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 3

Enter an integer to check for palindrome: 101

101 is a Palindrome.

STACK MENU

-
-
1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 3

Enter an integer to check for palindrome: 102

102 is not a Palindrome.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-
-

Enter your choice: 4

--- Demonstrating Stack Overflow ---

1 pushed onto the stack.

2 pushed onto the stack.

3 pushed onto the stack.

4 pushed onto the stack.

5 pushed onto the stack.

Stack Overflow! Cannot push 6.
Original stack restored.

--- Demonstrating Stack Underflow ---

Stack Underflow! Stack is empty.

Original stack preserved.

STACK MENU

-
1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-

Enter your choice: 4

--- Demonstrating Stack Overflow ---

1 pushed onto the stack.

2 pushed onto the stack.

3 pushed onto the stack.

4 pushed onto the stack.

5 pushed onto the stack.

Stack Overflow! Cannot push 6.

Original stack restored.

--- Demonstrating Stack Underflow ---

Stack Underflow! Stack is empty.

Original stack preserved.

STACK MENU

1. Push an Element
 2. Pop an Element
 3. Check Palindrome using Stack
 4. Demonstrate Overflow and Underflow
 5. Display Stack Status
 6. Exit
-

Enter your choice: 5

Stack elements from TOP to BOTTOM:
TOP -> 20 -> 10 -> BOTTOM

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

PROGRAM

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

#define MAX 5

// Structure to represent a Print Job
struct PrintJob
{
    int job_id;
    char document_name[50];
    int pages;
};

// Queue
struct PrintJob queue[MAX];
int front = -1;
int rear = -1;

// Function declarations
void addJob();
void processJob();
void displayJobs();
void countJobs();
void demonstrateOverflowUnderflow();

int main()
{
    int choice;
    do
    {
        printf("\n--- Printer Queue Simulation Menu ---\n");
        printf("1. Add a Print Job\n");
        printf("2. Process (Delete) a Print Job\n");
        printf("3. Display All Pending Print Jobs\n");
        printf("4. Display Number of Waiting Jobs\n");
        printf("5. Demonstrate Queue Overflow and Underflow\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
```

```
        case 1:
            addJob();
            break;
        case 2:
            processJob();
            break;

        case 3:
            displayJobs();
            break;

        case 4:
            countJobs();
            break;

        case 5:
            demonstrateOverflowUnderflow();
            break;

        case 6:
            printf("\nExiting Printer Queue simulation...\n");
            break;

        default:
            printf("\nInvalid choice! Please try again.\n");
    }
}
while (choice != 6);
return 0;
}
// a. Add a Print Job
void addJob()
{
    struct PrintJob newJob;
    // Check for overflow
    if (rear == MAX - 1)
    {
        printf("\nQueue Overflow! Printer queue is full.\n");
        printf("Cannot add more print jobs.\n");
        return;
    }
    printf("Enter Job ID: ");
    scanf("%d", &newJob.job_id);
    printf("Enter Document Name: ");
    scanf(" %[^\n]", newJob.document_name);
    printf("Enter Number of Pages: ");
    scanf("%d", &newJob.pages);
    // First element
    if (front == -1)
    {
        front = 0;
```

```

    }
    rear++;
    queue[rear] = newJob;
    printf("\nSuccess! Print Job '%s' (ID: %d) added to the queue.\n",
        newJob.document_name, newJob.job_id);
    }
// b. Process (Delete) a Print Job
void processJob()
{
    // Check for underflow
    if (front == -1 || front > rear)
    {
        printf("\nQueue Underflow! No pending print jobs to process.\n");
        return;
    }
    printf("\n--- Processing Print Job ---\n");
    printf("Job ID      : %d\n", queue[front].job_id);
    printf("Document Name: %s\n", queue[front].document_name);
    printf("Pages       : %d\n", queue[front].pages);
    printf("Status      : Printed and removed from queue.\n");
    front++;
    // Reset queue when empty
    if (front > rear)
    {
        front = -1;
        rear = -1;
    }
}
// c. Display all pending print jobs
void displayJobs()
{
    int i;
    if (front == -1 || front > rear)
    {
        printf("\nPrinter Queue is EMPTY. No pending jobs.\n");
        return;
    }
    printf("\n--- Pending Print Jobs in Queue ---\n");
    printf("%-10s %-25s %-10s\n",
        "Job ID", "Document Name", "Pages");
    printf("-----\n");
    for (i = front; i <= rear; i++)
    {
        printf("%-10d %-25s %-10d\n",
            queue[i].job_id,
            queue[i].document_name,
            queue[i].pages);
    }
}
// d. Count the number of jobs
void countJobs()

```

```
{
int count;
if (front == -1 || front > rear)
{
printf("\nNumber of jobs currently waiting: 0\n");
return;
}
count = rear - front + 1;
printf("\nNumber of print jobs currently waiting in the queue: %d\n",
count);
}
// e. Demonstrate Overflow and Underflow
void demonstrateOverflowUnderflow()
{
int demo_front = -1;
int demo_rear = -1;
int demo_queue[MAX];
int i;
printf("\n--- Demonstrating Queue Overflow and Underflow ---\n");
// Underflow
printf("\n1. Demonstrating Underflow:\n");
if (demo_front == -1)
{
printf("Queue is currently empty.\n");
printf("Attempting to process/dequeue a job...\n");
printf("-> Queue Underflow Encountered!\n");
printf("No jobs to process.\n");
}

// Overflow
printf("\n2. Demonstrating Overflow:\n");
printf("Filling the printer queue up to maximum size (MAX = %d)...\n",
MAX);

for (i = 0; i < MAX; i++)
{
if (demo_front == -1)
{
demo_front = 0;
}
demo_rear++;
demo_queue[demo_rear] = i + 101;
printf("Added Job ID: %d (Queue rear index: %d)\n",
demo_queue[demo_rear],
demo_rear);
}
// Try to add one more job
if (demo_rear == MAX - 1)
{
printf("\nQueue is now full (rear = %d).\n", demo_rear);
printf("Attempting to add one more job...\n");
}
```

```
        printf("-> Queue Overflow Encountered!\n");
        printf("Cannot add job, queue is full.\n");
    }
}
```

*****OUTPUT*****

--- Printer Queue Simulation Menu ---

1. Add a Print Job
2. Process (Delete) a Print Job
3. Display All Pending Print Jobs
4. Display Number of Waiting Jobs
5. Demonstrate Queue Overflow and Underflow
6. Exit

Enter your choice: 1

Enter Job ID: 101

Enter Document Name: Assignment

Enter Number of Pages: 10

Success! Print Job 'Assignment' (ID: 101) added to the queue.

--- Printer Queue Simulation Menu ---

1. Add a Print Job
2. Process (Delete) a Print Job
3. Display All Pending Print Jobs
4. Display Number of Waiting Jobs
5. Demonstrate Queue Overflow and Underflow
6. Exit

Enter your choice: 1

Enter Job ID: 102

Enter Document Name: Project

Enter Number of Pages: 20

Success! Print Job 'Project' (ID: 102) added to the queue.

--- Printer Queue Simulation Menu ---

Enter your choice: 3

--- Pending Print Jobs in Queue ---

Job ID	Document Name	Pages
101	Assignment	10
102	Project	20

--- Printer Queue Simulation Menu ---

Enter your choice: 4

Number of print jobs currently waiting in the queue: 2

--- Printer Queue Simulation Menu ---

Enter your choice: 2

--- Processing Print Job ---

Job ID 101

Document Name: Assignment

Pages 10

Status : Printed and removed from queue.

--- Printer Queue Simulation Menu ---

Enter your choice: 4

Number of print jobs currently waiting in the queue: 1

--- Printer Queue Simulation Menu ---

Enter your choice: 5

--- Demonstrating Queue Overflow and Underflow ---

1. Demonstrating Underflow:

Queue is currently empty.

Attempting to process/dequeue a job...

-> Queue Underflow Encountered!

no jobs to process.

2. Demonstrating Overflow:

Filling the printer queue up to maximum size (MAX = 5)...

Added Job ID: 101 (Queue rear index: 0)

Added Job ID: 102 (Queue rear index: 1)

Added Job ID: 103 (Queue rear index: 2)

Added Job ID: 104 (Queue rear index: 3)

Added Job ID: 105 (Queue rear index: 4)

Queue is now full (rear = 4).

Attempting to add one more job...

-> Queue Overflow Encountered!

Cannot add job, queue is full.

--- Printer Queue Simulation Menu ---

Enter your choice: 6

Exiting Printer Queue simulation...

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>

// Structure for Student Record
struct Student
{
    char usn[20];
    char name[50];
    char branch[20];
    int sem;
    char phone[15];
    struct Student *next;
};

// Global head pointer
struct Student *head = NULL;
// Function declarations
struct Student *createNode();
void createFront();
void displayAndCount();
void insertBegin();
void insertEnd();
void deleteBegin();
void deleteEnd();
void searchUSN();
int main()
{
    int choice;
    do
    {
        printf("\n--- Singly Linked List - Student Management System ---\n");
        printf("1. Create Linked List (Front Insertion)\n");
        printf("2. Display All Records & Count Nodes\n");
        printf("3. Insert at Beginning\n");
        printf("4. Insert at End\n");
        printf("5. Delete from Beginning\n");
        printf("6. Delete from End\n");
        printf("7. Search Student by USN\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
```

```
        case 1:
            createFront();
            break;
        case 2:
            displayAndCount();
            break;
        case 3:
            insertBegin();
            break;
        case 4:
            insertEnd();
            break;
        case 5:
            deleteBegin();
            break;
        case 6:
            deleteEnd();
            break;
        case 7:
            searchUSN();
            break;
        case 8:
            printf("\nExiting program...\n");
            break;
        default:
            printf("\nInvalid choice! Please try again.\n");
    }
} while (choice != 8);
// Free all remaining nodes
while (head != NULL)
{
    struct Student *temp = head;
    head = head->next;
    free(temp);
}
return 0;
}

// Function to create a new student node
struct Student *createNode()
{
    struct Student *newNode;
    newNode = (struct Student *)malloc(sizeof(struct Student));
    if (newNode == NULL)
    {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    printf("Enter USN: ");
    scanf("%19s", newNode->usn);
    printf("Enter Name: ");
    scanf(" %[^\n]", newNode->name);
```

```

printf("Enter Branch: ");
scanf("%19s", newNode->branch);
printf("Enter Semester: ");
scanf("%d", &newNode->sem);
printf("Enter Phone Number: ");
scanf("%14s", newNode->phone);
newNode->next = NULL;
return newNode;
}

// a. Create linked list using front insertion
void createFront()
{
    int n, i;
    struct Student *newNode;
    printf("\nEnter the number of student records to create: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nEnter details for Student %d:\n", i + 1);
        newNode = createNode();
        // Front insertion
        newNode->next = head;
        head = newNode;
        printf("Student record inserted at the front successfully.\n");
    }
}

// b. Display all records and count nodes
void displayAndCount()
{
    struct Student *temp;
    int count = 0;
    if (head == NULL)
    {
        printf("\nStudent list is EMPTY.\n");
        return;
    }
    temp = head;
    printf("\n--- Student Records ---\n");
    printf("%-15s %-20s %-12s %-5s %-15s\n",
           "USN", "Name", "Branch", "Sem", "Phone");
    printf("-----\n");
    while (temp != NULL)
    {
        printf("%-15s %-20s %-12s %-5s %-15s\n",
               temp->usn,
               temp->name,
               temp->branch,
               temp->sem,
               temp->phone);
        count++;
        temp = temp->next;
    }
}

```

```
    }
    printf("-----\n");
    printf("Total number of student records (nodes): %d\n", count);
}
// c. Insert at beginning
void insertBegin()
{
    struct Student *newNode;
    printf("\nEnter details for new student:\n");
    newNode = createNode();
    newNode->next = head;
    head = newNode;
    printf("\nStudent inserted at the beginning successfully.\n");
}
// c. Insert at end
void insertEnd()
{
    struct Student *newNode;
    struct Student *temp;
    printf("\nEnter details for new student:\n");
    newNode = createNode();
    if (head == NULL)
    {
        head = newNode;
    }
    else
    {
        temp = head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("\nStudent inserted at the end successfully.\n");
}
// c. Delete from beginning
void deleteBegin()
{
    struct Student *temp;

    if (head == NULL)
    {
        printf("\nList Underflow! No student records to delete.\n");
        return;
    }
    temp = head;
    head = head->next;
    printf("\nDeleted student record from beginning:\n");
    printf("USN : %s\n", temp->usn);
```

```
printf("Name : %s\n", temp->name);
free(temp);
}
// c. Delete from end
void deleteEnd()
{
struct Student *temp;
struct Student *prev;
if (head == NULL)
{
printf("\nList Underflow! No student records to delete.\n");
return;
}
// Only one node
if (head->next == NULL)
{
printf("\nDeleted student record from end:\n");
printf("USN : %s\n", head->usn);
printf("Name : %s\n", head->name);
free(head);
head = NULL;
return;
}
temp = head;
prev = NULL;
// Traverse to last node
while (temp->next != NULL)
{
prev = temp;
temp = temp->next;
}
prev->next = NULL;
printf("\nDeleted student record from end:\n");
printf("USN : %s\n", temp->usn);
printf("Name : %s\n", temp->name);
free(temp);
}
// d. Search student using USN
void searchUSN()
{
char target_usn[20];
struct Student *temp;
if (head == NULL)
{
printf("\nList is empty. Cannot search.\n");
return;
}
printf("\nEnter USN to search: ");
scanf("%19s", target_usn);
temp = head;
while (temp != NULL)
```

```
{
if (strcmp(temp->usn, target_usn) == 0)
{
    printf("\nStudent Record Found!\n");
    printf("-----\n");
    printf("USN   : %s\n", temp->usn);
    printf("Name  : %s\n", temp->name);
    printf("Branch : %s\n", temp->branch);
    printf("Sem   : %d\n", temp->sem);
    printf("Phone : %s\n", temp->phone);
    return;
}
temp = temp->next;
}
printf("\nStudent with USN '%s' not found in the list.\n",
        target_usn);
}
```

*****OUTPUT*****

--- Singly Linked List - Student Management System ---

1. Create Linked List (Front Insertion)
2. Display All Records & Count Nodes
3. Insert at Beginning
4. Insert at End
5. Delete from Beginning
6. Delete from End
7. Search Student by USN
8. Exit

Enter your choice: 1

Enter the number of student records to create: 3

Enter details for Student 1:

Enter USN: 1RV23CS001

Enter Name: Rahul

Enter Branch: CSE

Enter Semester: 3

Enter Phone Number: 9876543210

Student record inserted at the front successfully.

Enter details for Student 2:

Enter USN: 1RV23CS002

Enter Name: Priya

Enter Branch: CSE

DATA STRUCTURES LABORATORY**1BCSL306**

Enter Semester: 3

Enter Phone Number: 9876543211

Student record inserted at the front successfully.

Enter details for Student 3:

Enter USN: 1RV23CS003

Enter Name: Arjun

Enter Branch: ECE

Enter Semester: 3

Enter Phone Number: 9876543212

Student record inserted at the front successfully.

--- Singly Linked List - Student Management System ---

Enter your choice: 2

--- Student Records ---

USN	Name	Branch	Sem	Phone
-----	-----	-----	-----	-----
1RV23CS003	Arjun	ECE	3	9876543212
1RV23CS002	Priya	CSE	3	9876543211
1RV23CS001	Rahul	CSE	3	9876543210
-----	-----	-----	-----	-----

Total number of student records (nodes): 3

Enter your choice: 3

Enter details for new student:

Enter USN: 1RV23CS004

Enter Name: Sneha

Enter Branch: CSE

Enter Semester: 3

Enter Phone Number: 9876543213

Student inserted at the beginning successfully.

Enter your choice: 4

Enter details for new student:

Enter USN: 1RV23CS005

Enter Name: Kiran

Enter Branch: ECE

Enter Semester: 3

Enter Phone Number: 9876543214

Student inserted at the end successfully.

Enter your choice: 5

Deleted student record from beginning:

USN : 1RV23CS004

Name : Sneha

Enter your choice: 7

Enter USN to search: 1RV23CS002

Student Record Found!

USN : 1RV23CS002

Name : Priya

Branch : CSE

Sem : 3

Phone : 9876543211

Enter USN to search: 1RV23CS010

Student with USN '1RV23CS010' not found in the list.

5. Develop a Program in C for the following:
- Create a Binary Tree by inserting integer elements entered by the user in level order.
 - Implement the following functions:
 - createTree() – Create a binary tree using dynamic memory allocation.
 - preorder() – Display the elements using Preorder Traversal.
 - inorder() – Display the elements using Inorder Traversal.
 - postorder() – Display the elements using Postorder Traversal.
 - Provide a menu-driven interface with the following options:
 - Create Binary Tree
 - Display Preorder Traversal
 - Display Inorder Traversal
 - Display Postorder Traversal
 - Exit

```
#include <stdio.h>
#include <stdlib.h>
// Structure for Binary Tree Node
struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
};
// Structure for Queue Node
struct QueueNode
{
    struct Node *treeNode;
    struct QueueNode *next;
};
// Queue pointers
struct QueueNode *front = NULL;
struct QueueNode *rear = NULL;
// Global root
struct Node *root = NULL;
// Function declarations
struct Node *createNode(int value);
void enqueue(struct Node *node);
struct Node *dequeue();
void createTree();
void preorder(struct Node *curr);
void inorder(struct Node *curr);
void postorder(struct Node *curr);
void freeTree(struct Node *curr);
void clearQueue();
int main()
{
    int choice;
    do
    {
        printf("\n--- Binary Tree Operations Menu ---\n");
        printf("1. Create Binary Tree\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:
        createTree();
        break;
    case 2:
        printf("\nPreorder Traversal: ");
        if (root == NULL)
            printf("Tree is empty.");
        else
            preorder(root);
        printf("\n");
        break;
    case 3:
        printf("\nInorder Traversal: ");
        if (root == NULL)
            printf("Tree is empty.");
        else
            inorder(root);
        printf("\n");
        break;
    case 4:
        printf("\nPostorder Traversal: ");
        if (root == NULL)
            printf("Tree is empty.");
        else
            postorder(root);
        printf("\n");
        break;
    case 5:
        printf("\nExiting program...\n");
        // Free tree memory
        freeTree(root);

    // Clear queue memory
        clearQueue();
        break;
    default:
        printf("\nInvalid choice! Please try again.\n");
}
}
while (choice != 5);
return 0;
}

// Create a new tree node
```

```
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;
}
// Enqueue a tree node
void enqueue(struct Node *node)
{
    struct QueueNode *qNode;
    qNode = (struct QueueNode *)malloc(sizeof(struct QueueNode));
    if (qNode == NULL)
    {
        printf("Queue memory allocation failed!\n");
        exit(1);
    }
    qNode->treeNode = node;
    qNode->next = NULL;
    if (rear == NULL)
    {
        front = rear = qNode;
    }
    else
    {
        rear->next = qNode;
        rear = qNode;
    }
}
// Dequeue a tree node
struct Node *dequeue()
{
    struct QueueNode *temp;
    struct Node *node;
    if (front == NULL)
        return NULL;
    temp = front;
    node = temp->treeNode;
    front = front->next;
    if (front == NULL)
        rear = NULL;
    free(temp);
    return node;
}
```

```
// Create binary tree using level-order insertion
void createTree()
{
    int n, val, i;
    struct Node *newNode;
    struct Node *curr;
    printf("\nEnter the number of nodes to insert: ");
    scanf("%d", &n);
    if (n <= 0)
    {
        printf("Invalid number of nodes.\n");
        return;
    }
    // Free previous tree if it exists
    if (root != NULL)
    {
        freeTree(root);
        root = NULL;
    }
    // Clear old queue
    clearQueue();
    for (i = 0; i < n; i++)
    {
        printf("Enter value for node %d: ", i + 1);
        scanf("%d", &val);
        newNode = createNode(val);
        // First node becomes root
        if (root == NULL)
        {
            root = newNode;
            enqueue(newNode);
        }

        else
        {
            curr = front->treeNode;
            if (curr->left == NULL)
            {
                curr->left = newNode;
            }
            else
            {
                curr->right = newNode;

                // Parent has both children
                dequeue();
            }
            enqueue(newNode);
        }
    }
}
```

```
// Clear remaining queue nodes
clearQueue();
printf("\nBinary Tree created successfully using level-order insertion!\n");
}
// Preorder: Root -> Left -> Right
void preorder(struct Node *curr)
{
    if (curr != NULL)
    {
        printf("%d ", curr->data);
        preorder(curr->left);
        preorder(curr->right);
    }
}
// Inorder: Left -> Root -> Right
void inorder(struct Node *curr)
{
    if (curr != NULL)
    {
        inorder(curr->left);
        printf("%d ", curr->data);
        inorder(curr->right);
    }
}
// Postorder: Left -> Right -> Root
void postorder(struct Node *curr)
{
    if (curr != NULL)
    {
        postorder(curr->left);
        postorder(curr->right);
        printf("%d ", curr->data);
    }
}

// Free all tree nodes
void freeTree(struct Node *curr)
{
    if (curr != NULL)
    {
        freeTree(curr->left);
        freeTree(curr->right);
        free(curr);
    }
}

// Clear queue nodes
void clearQueue()
{
    struct QueueNode *temp;
    while (front != NULL)
```

```

    {
        temp = front;
        front = front->next;
        free(temp);
    }
    rear = NULL;
}

```

*****OUTPUT*****

--- Binary Tree Operations Menu ---

1. Create Binary Tree
2. Display Preorder Traversal
3. Display Inorder Traversal
4. Display Postorder Traversal
5. Exit

Enter your choice (1-5): 1

Enter the number of nodes to insert: 7

Enter value for node 1: 1

Enter value for node 2: 2

Enter value for node 3: 3

Enter value for node 4: 4

Enter value for node 5: 5

Enter value for node 6: 6

Enter value for node 7: 7

Binary Tree created successfully using level-order insertion!

```

      1
     / \
    2   3
   /\  /\
  4 5 6 7

```

Enter your choice (1-5): 2

Preorder Traversal: 1 2 4 5 3 6 7

Order Root->left->Right

inorder

Enter your choice (1-5): 3

Inorder Traversal: 4 2 5 1 6 3 7

Order Left → Root → Right

Post order

Enter your choice (1-5): 4

Postorder Traversal: 4 5 2 6 7 3 1

Order Left → Right → Root

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

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
int adj[MAX][MAX];
int visited[MAX];
int n;

// Function declarations
void createGraph();
void displayMatrix();
void DFS(int v);
void BFS(int start);

int main()
{
    int choice;
    int start;

    do
    {
        printf("\n--- Graph of Cities ---\n");
        printf("1. Create Adjacency Matrix\n");
        printf("2. Display Adjacency Matrix\n");
        printf("3. DFS - Reachable Nodes\n");
        printf("4. BFS - Reachable Nodes\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                createGraph();
                break;

            case 2:
                displayMatrix();
                break;

            case 3:
                if (n == 0)
                {
                    printf("\nGraph is not created.\n");
                }
                else
                {

```

```

printf("\nEnter starting city (1-%d): ", n);
scanf("%d", &start);
if (start < 1 || start > n)
{
printf("Invalid starting city!\n");
}
else
{
for (int i = 0; i < n; i++)
visited[i] = 0;
printf("\nCITIES reachable using DFS: ");
DFS(start - 1);
printf("\n");
}
}
break;

case 4:
if (n == 0)
{
printf("\nGraph is not created.\n");
}
else
{
printf("\nEnter starting city (1-%d): ", n);
scanf("%d", &start);

if (start < 1 || start > n)
{
printf("Invalid starting city!\n");
}
else
{
BFS(start - 1);
printf("\n");
}
}
break;
case 5:
printf("\nExiting program...\n");
break;
default:
printf("\nInvalid choice! Please try again.\n");
}

} while (choice != 5);
return 0;
}

// a. Create adjacency matrix
void createGraph()
{

```

```

    int i, j;
    printf("\nEnter the number of cities: ");
    scanf("%d", &n);
    if (n <= 0 || n > MAX)
    {
        printf("Invalid number of cities!\n");
        n = 0;
        return;
    }
    printf("\nEnter the adjacency matrix:\n");
    printf("Enter 1 if there is a direct path, otherwise enter 0.\n\n");

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("Is there a path from City %d to City %d? ", i + 1, j + 1);

            scanf("%d", &adj[i][j]);
        }
    }
    printf("\nAdjacency Matrix created successfully!\n");
}
// Display adjacency matrix
void displayMatrix()
{
    int i, j;
    if (n == 0)
    {
        printf("\nGraph is not created.\n");
        return;
    }
    printf("\n--- Adjacency Matrix ---\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("%d ", adj[i][j]);
        }
        printf("\n");
    }
}
// b. DFS traversal
void DFS(int v)
{
    int i; visited[v]
    = 1;
    printf("City%d ", v + 1);
    for (i = 0; i < n; i++)
    {

```

```

if (adj[v][i] == 1 && visited[i] == 0)
{
    DFS(i);
}
}
}
// BFS traversal
void BFS(int start)
{
    int queue[MAX];
    int front = 0;
    int rear = 0;
    int i;
    // Reset visited array
    for (i = 0; i < n; i++)
        visited[i] = 0;
    queue[rear++] = start;
    visited[start] = 1;
    printf("Cities reachable using BFS: ");
    while (front < rear)
    {
        int current = queue[front++];
        printf("City%d ", current + 1);
        for (i = 0; i < n; i++)
        {
            if (adj[current][i] == 1 && visited[i] == 0)
            {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }
    }
}
}

```

*****output*****

Suppose there are **5 cities** and the following directed graph is entered:

City1 → City2

City1 → City3

City2 → City4

City3 → City4

City4 → City5

The adjacency matrix is:

0 1 1 0 0

0 0 0 1 0

0 0 0 1 0

0 0 0 0 1
0 0 0 0 0

A sample execution:

--- Graph of Cities ---

1. Create Adjacency Matrix
2. Display Adjacency Matrix
3. DFS - Reachable Nodes
4. BFS - Reachable Nodes
5. Exit

Enter your choice: 1

Enter the number of cities: 5

Enter the adjacency matrix:

Enter 1 if there is a direct path, otherwise enter 0.

Is there a path from City 1 to City 1? 0

Is there a path from City 1 to City 2? 1

Is there a path from City 1 to City 3? 1

Is there a path from City 1 to City 4? 0

Is there a path from City 1 to City 5? 0

Is there a path from City 2 to City 1? 0

Is there a path from City 2 to City 2? 0

Is there a path from City 2 to City 3? 0

Is there a path from City 2 to City 4? 1

Is there a path from City 2 to City 5? 0

Is there a path from City 3 to City 1? 0

Is there a path from City 3 to City 2? 0

Is there a path from City 3 to City 3? 0

Is there a path from City 3 to City 4? 1

Is there a path from City 3 to City 5? 0

Is there a path from City 4 to City 1? 0

Is there a path from City 4 to City 2? 0

Is there a path from City 4 to City 3? 0

Is there a path from City 4 to City 4? 0

Is there a path from City 4 to City 5? 1

Is there a path from City 5 to City 1? 0

Is there a path from City 5 to City 2? 0

Is there a path from City 5 to City 3? 0

Is there a path from City 5 to City 4? 0

Is there a path from City 5 to City 5? 0

Adjacency Matrix created successfully!

Enter your choice: 2

--- Adjacency Matrix ---

0 1 1 0 0

0 0 0 1 0

0 0 0 1 0

0 0 0 0 1

0 0 0 0 0

Enter your choice: 3

Enter starting city (1-5): 1

Cities reachable using DFS: City1 City2 City4 City5 City3

Enter your choice: 4

Enter starting city (1-5): 1

Cities reachable using BFS: City1 City2 City3 City4 City5

7. Develop a C program to:

- Read two sparse matrices of the same order.
- Convert both into 3-tuple representation.
- Perform matrix addition.
- Display the resultant matrix in both normal matrix form and 3-tuple form.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

// Structure to represent a term in 3-tuple format
struct SparseTuple {
    int row;
    int col;
    int value;
};

// Function declarations
void readSparseMatrix(int matrix[MAX][MAX], int rows, int cols);
void convertToTuple(int matrix[MAX][MAX], int rows, int cols, struct SparseTuple tuple[]);
void addTuple(struct SparseTuple t1[], struct SparseTuple t2[], struct SparseTuple result[]);
void displayTuple(struct SparseTuple tuple[]);
void displayNormalMatrix(struct SparseTuple tuple[]);

int main() {
    int rows, cols;
    int matrix1[MAX][MAX], matrix2[MAX][MAX];
    struct SparseTuple t1[MAX], t2[MAX], result[MAX];
    printf("Enter the number of rows and columns for the sparse matrices: ");
    scanf("%d %d", &rows, &cols);
    // a. Read two sparse matrices of the same order
    printf("\nEnter elements for Matrix 1 (%dx%d):\n", rows, cols);
    readSparseMatrix(matrix1, rows, cols);
    printf("\nEnter elements for Matrix 2 (%dx%d):\n", rows, cols);
    readSparseMatrix(matrix2, rows, cols);
    // b. Convert both into 3-tuple representation
    t1[0].row = rows;
    t1[0].col = cols;
    convertToTuple(matrix1, rows, cols, t1);
    t2[0].row = rows;
    t2[0].col = cols;
    convertToTuple(matrix2, rows, cols, t2);
    // Display the initial 3-tuple representations
    printf("\n--- 3-Tuple Representation of Matrix 1 ---");
    displayTuple(t1);
    printf("\n--- 3-Tuple Representation of Matrix 2 ---");
    displayTuple(t2);
    // c. Perform matrix addition using 3-tuple representation
    addTuple(t1, t2, result);
    // d. Display the resultant matrix in both 3-tuple form and normal matrix form
    printf("\n--- Resultant Matrix (3-Tuple Form) ---");
```

```

displayTuple(result);
printf("\n--- Resultant Matrix (Normal Form) ---");
displayNormalMatrix(result);
return 0;
}

// Function to read a standard matrix from user input
void readSparseMatrix(int matrix[MAX][MAX], int rows, int cols)
{
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++)
        {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &matrix[i][j]);
        }
    }
}

// Function to convert a normal matrix into 3-tuple format
void convertToTuple(int matrix[MAX][MAX], int rows, int cols, struct SparseTuple tuple[])
{
    int k = 1; // Index for non-zero elements in tuple array
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            if (matrix[i][j] != 0)
            {
                tuple[k].row = i;
                tuple[k].col = j;
                tuple[k].value = matrix[i][j];
                k++;
            }
        }
    }

    tuple[0].value = k - 1; // Total number of non-zero values stored at index 0
}

// Function to add two sparse matrices given in 3-tuple form
void addTuple(struct SparseTuple t1[], struct SparseTuple t2[], struct SparseTuple result[]) {
    // Check if matrix dimensions match
    if (t1[0].row != t2[0].row || t1[0].col != t2[0].col)
    {
        printf("\nError: Matrix orders must be the same for addition!\n");
        result[0].row = 0;
        result[0].col = 0;
        result[0].value = 0;
        return;
    }

    result[0].row = t1[0].row;
    result[0].col = t1[0].col;
    int i = 1, j = 1, k = 1;
    while (i <= t1[0].value && j <= t2[0].value) {
        // Compare row indices first

        if (t1[i].row < t2[j].row || (t1[i].row == t2[j].row && t1[i].col < t2[j].col))
        {
            result[k] = t1[i];

```

```

        i++;
        k++;
    }
    // Compare column indices if rows are equal
    else if (t2[j].row < t1[i].row || (t2[j].row == t1[i].row && t2[j].col < t1[i].col))
    {
        result[k] = t2[j];
        j++;
        k++;
    }
    // Both row and column match, add their values
    else {
        int sum = t1[i].value + t2[j].value;
        if (sum != 0) {
            result[k].row = t1[i].row;
            result[k].col = t1[i].col;
            result[k].value = sum;
            k++;
        }
        i++;
        j++;
    }
}

// Copy remaining elements from t1, if any
while (i <= t1[0].value) {
    result[k++] = t1[i++];
}
// Copy remaining elements from t2, if any
while (j <= t2[0].value) {
    result[k++] = t2[j++];
}
result[0].value = k - 1; // Set total non-zero count
}

// Function to display a 3-tuple representation table
void displayTuple(struct SparseTuple tuple[])
{
    int total_elements = tuple[0].value;
    printf("\n%-8s %-8s %-8s\n", "Row", "Column", "Value");
    printf("-----\n");
    printf("%-8d %-8d %-8d (Header)\n", tuple[0].row, tuple[0].col, tuple[0].value);
    for (int i = 1; i <= total_elements; i++)
    {
        printf("%-8d %-8d %-8d\n", tuple[i].row, tuple[i].col, tuple[i].value);
    }
}

// Function to display the 3-tuple format back into normal matrix form
void displayNormalMatrix(struct SparseTuple tuple[]) {
    int rows = tuple[0].row;

```

```

int cols = tuple[0].col;
int total = tuple[0].value;
int k = 1;
printf("\n");
for (int i = 0; i < rows; i++)
{ for (int j = 0; j < cols; j++) {
if (k <= total && tuple[k].row == i && tuple[k].col == j)
{ printf("%-4d ", tuple[k].value);
k++;
}
else
{
printf("%-4d ", 0);
}
}
}

printf("\n");
}
}

```

*****OUTPUT*****

Enter the number of rows and columns for the sparse matrices: 3 3

Matrix 1:

```

1 0 0
0 2 0
0 0 3

```

Matrix 2:

```

4 0 0
0 5 0
0 0 6

```

--- 3-Tuple Representation of Matrix 1 ---

Row	Column	Value	
3	3	3	(Header)
0	0	1	
1	1	2	
2	2	3	

--- 3-Tuple Representation of Matrix 2 ---

Row	Column	Value	
3	3	3	(Header)
0	0	4	

1	1	5
2	2	6

--- Resultant Matrix (3-Tuple Form) ---

Row	Column	Value

3	3	3
0	0	5
1	1	7
2	2	9

--- Resultant Matrix (Normal Form) ---

5	0	0
0	7	0
0	0	9

8. 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 <string.h>
#include <ctype.h>

#define MAX 100
// Stack structure for operators
char stack[MAX];
int top = -1;
// Stack helper functions
void push(char item)
{
    if (top >= MAX - 1)
    {
        printf("Stack Overflow\n");
    }
    else
    {
        stack[++top] = item;
    }
}
char pop()
{
    if (top < 0)
    {
        return '\0';
    }
    else
    {
        return stack[top--];
    }
}

char peek()
{
    if (top < 0)
    {
        return '\0';
    }
    else
    {
        return stack[top];
    }
}
```

```
// Function to determine precedence of operators
```

```
int precedence(char symbol)
{
    switch (symbol)
    {
        case '^':
            return 3;

        case '*':
        case '/':
        case '%':
            return 2;

        case '+':
        case '-':
            return 1;

        default:
            return 0;
    }
}
```

```
// Function to check associativity
```

```
// '^' is Right-to-Left associative
```

```
// Other operators are Left-to-Right associative
```

```
int isRightAssociative(char symbol)
{
    if(symbol == '^')
        return 1;
    return 0;
}
```

```
// Function to convert infix expression to postfix
```

```
void infixToPostfix(char infix[], char postfix[])
{
    int i, j = 0;
    char symbol;
    // Reset stack
    top = -1;
    for (i = 0; infix[i] != '\0'; i++)
    {
        symbol = infix[i];

        // Skip whitespace
        if (symbol == ' ' || symbol == '\t')
        {
            continue;
        }
        // If operand, add it to postfix
        if (isalnum((unsigned char)symbol))
        {

```

```

postfix[j++] = symbol;
}
// If opening parenthesis, push it
else if (symbol == '(')
{
    push(symbol);
}
// If closing parenthesis
else if (symbol == ')')
{
    while (top != -1 && peek() != '(')
    {
        postfix[j++] = pop();
    }

    // Remove '(' from stack
    if (top != -1 && peek() == '(')
    {
        pop();
    }
}
// If operator
else
{
    while (top != -1 &&
        peek() != '(' &&
        (precedence(peek()) > precedence(symbol) ||
        (precedence(peek()) == precedence(symbol) &&
        !isRightAssociative(symbol))))
    {
        postfix[j++] = pop();
    }
    push(symbol);
}

// Pop remaining operators
while (top != -1)
{
    postfix[j++] = pop();
}
// Null terminate postfix string
postfix[j] = '\0';
}

// Function to run test cases
void runTestCases()
{
    char *testCases[][2] =
    {
        {"A+B*C", "ABC*+"},

```

```

{"(A+B)*C", "AB+C*"},
{"A^B^C", "ABC^^"},
{"A*(B+C)/D", "ABC+*D/"},
{"a+b*(c-d)/e", "abcd-*e/+"},
{"A+B*C-D/E", "ABC*+DE/-"},
{"(A+B)*(C-D)", "AB+CD-*"}
};
int numTests = sizeof(testCases) / sizeof(testCases[0]);
printf("\n--- Running Validation Test Cases ---\n");
for (int i = 0; i < numTests; i++)
{
    char postfix[MAX];
    infixToPostfix(testCases[i][0], postfix);
    printf("\nTest %d:\n", i + 1);
    printf("Infix   : %s\n", testCases[i][0]);
    printf("Expected : %s\n", testCases[i][1]);
    printf("Postfix  : %s\n", postfix);
    if (strcmp(postfix, testCases[i][1]) == 0)
    {
        printf("Result   : PASS\n");
    }
    else
    {
        printf("Result   : FAIL\n");
    }
}
}
// Main function
int main()
{
    int choice;
    char infix[MAX];
    char postfix[MAX];
    do
    {
        printf("\n===== \n");
        printf("    EXPRESSION CONVERSION TOOL\n");
        printf("    INFIX TO POSTFIX\n");
        printf("===== \n");
        printf("1. Convert Infix Expression to Postfix\n");
        printf("2. Run Built-in Validation Test Cases\n");
        printf("3. Exit\n");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        // Clear newline from input buffer
        getchar();
        switch (choice)
        {
            case 1:
                printf("\nEnter a valid infix expression: ");
                fgets(infix, sizeof(infix), stdin);

```

```
// Remove trailing newline
infix[strcspn(infix, "\n")] = '\0';
infixToPostfix(infix, postfix);
printf("\nPostfix Expression: %s\n", postfix);
break;
case 2:
runTestCases();
break;
case 3:
printf("\nExiting tool...\n");
break;
default:
printf("\nInvalid choice! Please try again.\n");
}
}
while (choice != 3);
return 0;
}
```

*****OUTPUT*****

EXPRESSION CONVERSION TOOL
INFIX TO POSTFIX

1. Convert Infix Expression to Postfix
2. Run Built-in Validation Test Cases
3. Exit

:

Enter your choice: 1

Enter a valid infix expression: A+B*C

Postfix Expression: ABC*+

Enter your choice: 2

--- Running Validation Test Cases ---

Test 1:

Infix : A+B*C

Expected : ABC*+

Postfix : ABC*+

Result : PASS

Test 2:

Infix : (A+B)*C

Expected : AB+C*

Postfix : AB+C*

Result : PASS

Test 3:

Infix : A^B^C

Expected : ABC^^

Postfix : ABC^^

Result : PASS

Test 4:

Infix : $A*(B+C)/D$

Expected : ABC+*D/

Postfix : ABC+*D/

Result : PASS

Test 5:

Infix : $a+b*(c-d)/e$

Expected : abcd-*e/+

Postfix : abcd-*e/+

Result : PASS

9. 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>
#define MAX 5
float cqueue[MAX];
int front = -1;
int rear = -1;
/* Function declarations */
void insertElement();
void deleteElement();
void displayQueue();
void demonstrateOverflowUnderflow();
int main()
{
    int choice;
    do
    {
        printf("\n===== CIRCULAR QUEUE MENU =====\n");
        printf("1. Insert an Element\n");
        printf("2. Delete an Element\n");
        printf("3. Display Queue Contents\n");
        printf("4. Demonstrate Overflow and Underflow\n");
        printf("5. Exit\n");
        printf("===== \n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                insertElement();
                break;
            case 2:
                deleteElement();
                break;
            case 3:
                displayQueue();
                break;
            case 4:
                demonstrateOverflowUnderflow();
                break;
            case 5:
                printf("\nExiting the program...\n");
                break;
        }
    }
}
```

```
default:
    printf("\nInvalid choice! Please try again.\n");
}
} while (choice != 5);
return 0;
}
```

```
/* a. Insert an element */
```

```
void insertElement()
```

```
{
    float element;
    /* Check for overflow */
    if ((front == 0 && rear == MAX - 1) ||
        (front == rear + 1))
    {
        printf("\nCircular Queue Overflow!\n");
        printf("Queue is full. Cannot insert element.\n");
        return;
    }
    printf("Enter the element to insert: ");
    scanf("%f", &element);
    /* Queue is initially empty */
    if (front == -1)
    {
        front = 0;
        rear = 0;
    }
    /* Wrap rear to beginning */
    else if (rear == MAX - 1)
    {
        rear = 0;
    }
    else
    {
        rear++;
    }
    cqueue[rear] = element;
    printf("Successfully inserted %.2f\n", element);
}
```

```
/* b. Delete an element */
```

```
void deleteElement()
```

```
{

    float deletedItem;
    /* Check for underflow */
    if (front == -1)
    {
        printf("\nCircular Queue Underflow!\n");
        printf("Queue is empty. Cannot delete element.\n");
        return;
    }
}
```

```
}
```

```
deletedItem = cqueue[front];
printf("Deleted element: %.2f\n", deletedItem);
/* Only one element was present */
if (front == rear)
{
    front = -1;
    rear = -1;
}
/* Wrap front to beginning */
else if (front == MAX - 1)
{
    front = 0;
}
else
{
    front++;
}
}
```

```
/* c. Display the circular queue */
void displayQueue()
{
    int i;
    if (front == -1)
    {
        printf("\nCircular Queue is EMPTY.\n");
        return;
    }
    printf("\n===== CIRCULAR QUEUE =====\n");
    printf("Front = %d\n", front);
    printf("Rear = %d\n", rear);
    printf("Elements: ");
    /* No wrap-around */
    if (rear >= front)
    {
        for (i = front; i <= rear; i++)
        {
            printf("%.2f ", cqueue[i]);
        }
    }
    /* Wrap-around case */
    else
    {
        for (i = front; i < MAX; i++)
        {
            printf("%.2f ", cqueue[i]);
        }
        for (i = 0; i <= rear; i++)
        {
            printf("%.2f ", cqueue[i]);
        }
    }
}
```

```

    }

    }
    printf("\n===== \n");
}
/* d. Demonstrate overflow and underflow */
void demonstrateOverflowUnderflow()
{
    float demo_q[MAX];
    int d_front = -1;
    int d_rear = -1;
    int i;
    printf("\n===== OVERFLOW AND UNDERFLOW ===== \n");
    /* Demonstrate underflow */
    printf("\n1. UNDERFLOW: \n");
    if (d_front == -1)
    {
        printf("Queue is empty. \n");
        printf("Attempting to delete an element... \n");
        printf("Circular Queue Underflow! \n");
        printf("No elements available for deletion. \n");
    }
    /* Demonstrate overflow */
    printf("\n2. OVERFLOW: \n");
    printf("Filling queue with %d elements... \n", MAX);
    for (i = 0; i < MAX; i++)
    {
        if (d_front == -1)
        {
            d_front = 0;
        }
        d_rear = (d_rear + 1) % MAX;
        demo_q[d_rear] = (i + 1) * 10.5;
        printf("Inserted %.2f (Front = %d, Rear = %d) \n",
            demo_q[d_rear], d_front, d_rear);
    }
    /* Queue is now full */
    if ((d_front == 0 && d_rear == MAX - 1) ||
        (d_front == d_rear + 1))
    {
        printf("\nQueue is FULL. \n");
        printf("Attempting to insert one more element... \n");
        printf("Circular Queue Overflow! \n");
        printf("Cannot insert because queue is full. \n");
    } printf("\n===== \n");
}

```

*****OUTPUT*****

===== CIRCULAR QUEUE MENU =====

1. Insert an Element
2. Delete an Element
3. Display Queue Contents
4. Demonstrate Overflow and Underflow
5. Exit

=====

Enter your choice: 1
 Enter the element to insert: 10.5
 Successfully inserted 10.50

===== CIRCULAR QUEUE MENU =====

1. Insert an Element
2. Delete an Element
3. Display Queue Contents
4. Demonstrate Overflow and Underflow
5. Exit

=====

Enter your choice: 1
 Enter the element to insert: 20.5
 Successfully inserted 20.50

===== CIRCULAR QUEUE MENU =====

1. Insert an Element
2. Delete an Element
3. Display Queue Contents
4. Demonstrate Overflow and Underflow
5. Exit

=====

Enter your choice: 1
 Enter the element to insert: 30.5
 Successfully inserted 30.50

Enter your choice: 3

===== CIRCULAR QUEUE =====

Front = 0
 Rear = 2
 Elements: 10.50 20.50 30.50

Enter your choice: 2
 Deleted element: 10.50

===== CIRCULAR QUEUE =====

Front = 1
 Rear = 2
 Elements: 20.50 30.50

===== OVERFLOW AND UNDERFLOW =====

1. UNDERFLOW:

Queue is empty.
Attempting to delete an element...
Circular Queue Underflow!
No elements available for deletion.

2. OVERFLOW:

Filling queue with 5 elements...
Inserted 10.50 (Front = 0, Rear = 0)
Inserted 21.00 (Front = 0, Rear = 1)
Inserted 31.50 (Front = 0, Rear = 2)
Inserted 42.00 (Front = 0, Rear = 3)
Inserted 52.50 (Front = 0, Rear = 4)

Queue is FULL.
Attempting to insert one more element...
Circular Queue Overflow!
Cannot insert because queue is full.

10. 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>
/* Define the Employee structure */
struct Employee
{
    char ssn[20];
    char name[50];
    char dept[20];
    char designation[30];
    float sal;
    char phno[15];
    struct Employee *prev;
    struct Employee *next;
};
/* Global pointers */
struct Employee *head = NULL;
struct Employee *tail = NULL;
/* Function declarations */
struct Employee *createNode();
void createDLL();
void displayAndCount();
void insertEnd();
void deleteEnd();
void insertFront();
void deleteFront();
void demonstrateDeque();
void freeList();
/* Create a new employee node */
struct Employee *createNode()
{
    struct Employee *newNode;
    newNode = (struct Employee *)malloc(sizeof(struct Employee));
    if (newNode == NULL)
    {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    printf("Enter SSN: ");
    scanf("%19s", newNode->ssn);
```

```
printf("Enter Name: ");
scanf(" %49[^\n]", newNode->name);

printf("Enter Department: ");
scanf("%19s", newNode->dept);

printf("Enter Designation: ");
scanf(" %29[^\n]", newNode->designation);

printf("Enter Salary: ");
scanf("%f", &newNode->sal);

printf("Enter Phone Number: ");
scanf("%14s", newNode->phno);

newNode->prev = NULL;
newNode->next = NULL;

return newNode;
}

/* a. Create DLL of N employees using end insertion */
void createDLL()
{
    int n, i;
    struct Employee *newNode;

    printf("\nEnter the number of employees: ");
    scanf("%d", &n);

    if (n <= 0)
    {
        printf("Number of employees must be greater than 0.\n");
        return;
    }
    /* Clear existing list before creating a new one */
    freeList();

    for (i = 0; i < n; i++)
    {
        printf("\nEnter details for Employee %d:\n", i + 1);
        newNode = createNode();
        if (head == NULL)
        {
            head = tail = newNode;
        }
        else
        {
            tail->next = newNode;
            newNode->prev = tail;
        }
    }
}
```

```

tail = newNode;

    }
}

printf("\nDLL created successfully with %d employee(s).\n", n);
}

/* b. Display DLL and count number of nodes */
void displayAndCount()
{
    struct Employee *temp;
    int count = 0;
    if (head == NULL)
    {
        printf("\nDoubly Linked List is EMPTY.\n");
        return;
    }
    temp = head;
    printf("\n===== EMPLOYEE RECORDS =====\n");
    printf("%-12s %-20s %-12s %-18s %-10s %-15s\n",
        "SSN", "Name", "Department",
        "Designation", "Salary", "Phone");

    printf("-----\n");
    while (temp != NULL)
    {
        printf("%-12s %-20s %-12s %-18s %-10.2f %-15s\n",
            temp->ssn,
            temp->name,
            temp->dept,
            temp->designation,
            temp->sal,
            temp->phno);
        count++;
        temp = temp->next;
    }
    printf("-----\n");
    printf("Total number of nodes (employees): %d\n", count);
}

/* c. Insert at End */
void insertEnd()
{
    struct Employee *newNode;
    printf("\nEnter details of employee to insert at END:\n");
    newNode = createNode();
    if (head == NULL)
    {
        head = tail = newNode;
    }
}

```

```
else
{

    tail->next = newNode;
    newNode->prev = tail;
    tail = newNode;
}
printf("\nEmployee inserted at END successfully.\n");
}

/* c. Delete from End */
void deleteEnd()
{
    struct Employee *temp;
    if (head == NULL)
    {
        printf("\nList Underflow! No employee records to delete.\n");
        return;
    }
    temp = tail;
    printf("\nDeleted employee from END:\n");
    printf("SSN : %s\n", temp->ssn);
    printf("Name : %s\n", temp->name);
    if (head == tail)
    {
        head = NULL;
        tail = NULL;
    }
    else
    {
        tail = tail->prev;
        tail->next = NULL;
    }
    free(temp);
    printf("Employee deleted from END successfully.\n");
}

/* d. Insert at Front */
void insertFront()
{
    struct Employee *newNode;
    printf("\nEnter details of employee to insert at FRONT:\n");
    newNode = createNode();
    if (head == NULL)
    {
        head = tail = newNode;
    }
    else
    {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}
```

```

printf("\nEmployee inserted at FRONT successfully.\n");
}

/* d. Delete from Front */
void deleteFront()
{
    struct Employee *temp;
    if (head == NULL)
    {
        printf("\nList Underflow! No employee records to delete.\n");
        return;
    }
    temp = head;
    printf("\nDeleted employee from FRONT:\n");
    printf("SSN : %s\n", temp->:ssn);
    printf("Name : %s\n", temp->name);
    if (head == tail)
    {
        head = NULL;
        tail = NULL;
    }
    else
    {
        head = head->next;
        head->prev = NULL;
    }
    free(temp);
    printf("Employee deleted from FRONT successfully.\n");
}

/* e. Demonstrate DLL as Double Ended Queue */
void demonstrateDeque()
{ printf("\n===== \n");
; printf("    DOUBLE ENDED QUEUE (DEQUE) USING DLL\n");
printf("===== \n");
printf("\nA Deque allows insertion and deletion at both ends.\n");
printf("\nFRONT OPERATIONS:\n");
printf("1. Insert at Front\n");
printf("2. Delete from Front\n");
printf("\nREAR OPERATIONS:\n");
printf("1. Insert at End\n");
printf("2. Delete from End\n");
printf("\nDLL Deque Mapping:\n");
printf("Front of Deque -> Head of DLL\n");
printf("Rear of Deque -> Tail of DLL\n");
printf("\nTime Complexity:\n");
printf("Insertion at Front : O(1)\n");
printf("Deletion at Front : O(1)\n");
printf("Insertion at Rear : O(1)\n");
printf("Deletion at Rear : O(1)\n");
}

```

```

printf("\n=====\\n");
}

/* Free all nodes */

void freeList()
{
    struct Employee *temp;
    while (head != NULL)
    {
        temp = head;
        head = head->next;
        free(temp);
    }
    tail = NULL;
}

/* Main function */
int main()
{
    int choice;
    int subChoice;
    do
    {
        printf("\n=====\\n");
        printf("    DOUBLY LINKED LIST - EMPLOYEE DATA\\n");
        printf("=====\\n");
        printf("1. Create DLL of N Employees\\n");
        printf("2. Display DLL and Count Nodes\\n");
        printf("3. Insert/Delete at End\\n");
        printf("4. Insert/Delete at Front\\n");
        printf("5. Demonstrate DLL as Double Ended Queue\\n");
        printf("6. Exit\\n");
        printf("=====\\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                createDLL();
                break;
            case 2:
                displayAndCount();
                break;
            case 3:
                printf("\n--- END OPERATIONS ---\\n");
                printf("1. Insert at End\\n");
                printf("2. Delete from End\\n");
                printf("Enter your choice: ");
                scanf("%d", &subChoice);
                if (subChoice == 1)
                {

```

```
        insertEnd();
    }
    else if (subChoice == 2)
    {
        deleteEnd();
    }
    else
    {
        printf("Invalid choice!\n");
    }
    break;
case 4:
    printf("\n--- FRONT OPERATIONS ---\n");
    printf("1. Insert at Front\n");
    printf("2. Delete from Front\n");
    printf("Enter your choice: ");
    scanf("%d", &subChoice);
    if (subChoice == 1)
    {
        insertFront();
    }
    else if (subChoice == 2)
    {
        deleteFront();
    }
    else
    {
        printf("Invalid choice!\n");
    }
    break;
case 5:
    demonstrateDeque();
    break;
case 6:
    printf("\nExiting program...\n");
    freeList();
    break;
default:
    printf("\nInvalid choice! Please try again.\n");
}
}
while (choice != 6);
return 0;
}
```

*****OUTPUT*****

=====

DOUBLY LINKED LIST - EMPLOYEE DATA

- =====
1. Create DLL of N Employees
 2. Display DLL and Count Nodes
 3. Insert/Delete at End
 4. Insert/Delete at Front
 5. Demonstrate DLL as Double Ended Queue
 6. Exit
- =====

Enter your choice: 1

Enter the number of employees: 2

Enter details for Employee 1:

Enter SSN: 101

Enter Name: Rahul Kumar

Enter Department: CSE

Enter Designation: Developer

Enter Salary: 45000

Enter Phone Number: 9876543210

Enter details for Employee 2:

Enter SSN: 102

Enter Name: Priya Sharma

Enter Department: ECE

Enter Designation: Engineer

Enter Salary: 50000

Enter Phone Number: 9876501234

DLL created successfully with 2 employee(s).

Enter your choice: 2

===== EMPLOYEE RECORDS =====

SSN	Name	Department	Designation	Salary	Phone
101	Rahul Kumar	CSE	Developer	45000.00	9876543210
102	Priya Sharma	ECE	Engineer	50000.00	9876501234

Total number of nodes (employees): 2

11. 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>
/* Structure for a BST node */
struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
};

/* Function declarations */
struct Node *createNode(int value);
struct Node *insert(struct Node *root, int value);
void inorder(struct Node *root);
void preorder(struct Node *root);
void postorder(struct Node *root);
struct Node *search(struct Node *root, int key);
void freeTree(struct Node *root);
int main()
{
    struct Node *root = NULL;
    int choice;
    int n, value, key;
    int i;
    do
    {
        printf("\n===== \n");
        printf("    BINARY SEARCH TREE (BST)\n");
        printf("===== \n");
        printf("1. Create BST of N Integers\n");
        printf("2. Traverse BST\n");
        printf("3. Search for an Element\n");
        printf("4. Exit\n");
        printf("===== \n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            /* a. Create BST */
            case 1:
                /* Clear existing tree */
                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;
/* b. Traversals */
case 2:
    if (root == NULL)
    {
        printf("\nBST is empty. Please create the BST first.\n");
        break;
    }
    printf("\nInorder Traversal : ");
    inorder(root);
    printf("\nPreorder Traversal : ");
    preorder(root);
    printf("\nPostorder Traversal: ");
    postorder(root);
    printf("\n");
    break;
/* c.Search */
case 3:
    if (root == NULL)
    {
        printf("\nBST is empty. Please create the BST first.\n");
        break;
    }
    printf("\nEnter the element to search: ");
    scanf("%d", &key);
    if (search(root, key) != NULL)
    {
        printf("Element %d is FOUND in the BST.\n", key);
    }
    else
    {
        printf("Element %d is NOT FOUND in the BST.\n", key);
    }
    break;
/* d. Exit */
case 4:
```

```
    printf("\nExiting program...\n");
    freeTree(root);
    break;
default:
    printf("\nInvalid choice! Please try again.\n");
    }
    }
while (choice != 4);
return 0;
}
```

```
/* Create a new BST node */
```

```
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;
}
```

```
/* Insert an element into BST */
```

```
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 value %d ignored.\n", value);
    }
    return root;
}
```

```
/* Inorder Traversal: Left -> Root -> Right */
```

```
void inorder(struct Node *root)
{

```

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

/* Preorder Traversal: Root -> Left -> Right */
void preorder(struct Node *root)
{
    if (root != NULL)
    {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

/* Postorder Traversal: Left -> Right -> Root */
void postorder(struct Node *root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

/* Search for an element in BST */
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);
    }
}

/* Free all nodes of BST */
void freeTree(struct Node *root)
{
    if (root != NULL)
    {
        freeTree(root->left);
```

```
freeTree(root->right);
free(root);
}
}
```

*****OUTPUT*****

50 30 70 20 40 60 80

BINARY SEARCH TREE (BST)

1. Create BST of N Integers
 2. Traverse BST
 3. Search for an Element
 4. Exit
-
-

Enter your choice: 1

Enter the number of integers: 7

Enter 7 integers:

50 30 70 20 40 60 80

BST created successfully.

Enter your choice: 2

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

Enter your choice: 3

Enter the element to search: 60

Element 60 is FOUND in the BST.

Enter your choice: 3

Enter the element to search: 100

Element 100 is NOT FOUND in the BST.

```

      50
     /  \
    30   70
   / \  / \
  20 40 60 80
```

Inorder : Left $\rightarrow$ Root $\rightarrow$ Right
: 20 30 40 50 60 70 80

Preorder : Root $\rightarrow$ Left $\rightarrow$ Right
: 50 30 20 40 70 60 80

Postorder : Left $\rightarrow$ Right $\rightarrow$ Root
: 20 40 30 60 80 70 50

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

- Create a hash table to store employee records using Employee ID as the key.
- Insert employee records into the hash table using the Hash function $H: K \rightarrow L$ as $H(K) = K \bmod m$
- Resolve collisions using Linear Probing.
- 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>
#include <string.h>

#define MAX 100
/* Employee structure */
struct Employee
{
    int empID;
    char name[50];
    char department[30];
    float salary;
};

/* Hash table structure */
struct Employee hashTable[MAX];

/* Occupied array
   0 = Empty
   1 = Occupied
*/
int occupied[MAX];

/* Size of hash table */
int tableSize;

/* Function declarations */
int hashFunction(int key);
void initializeTable();
void insertEmployee();
void searchEmployee();
void displayTable();
/* Hash function:  $H(K) = K \bmod m$  */
int hashFunction(int key)
{
    return key % tableSize;
}
/* Initialize hash table */
void initializeTable()
{
    int i;
    for (i = 0; i < tableSize; i++)
```

```
{
occupied[i] = 0;
}
}

/* Insert employee using Linear Probing */
void insertEmployee()
{
    struct Employee emp;
    int index;
    int startIndex;
    int i;
    printf("\nEnter Employee ID: ");
    scanf("%d", &emp.empID);
    /* Calculate hash index */
    index = hashFunction(emp.empID);
    startIndex = index;
    /* Linear probing */
    while (occupied[index] == 1)
    {
        /* Check for duplicate Employee ID */
        if (hashTable[index].empID == emp.empID)
        {
            printf("\nEmployee ID %d already exists!\n", emp.empID);
            return;
        }
        index = (index + 1) % tableSize;
        /* Hash table is full */
        if (index == startIndex)
        {
            printf("\nHash Table Overflow!\n");
            printf("No space available for insertion.\n");
            return;
        }
    }
    /* Read employee details */
    printf("Enter Employee Name: ");
    scanf(" %49[^\n]", emp.name);
    printf("Enter Department: ");
    scanf(" %29[^\n]", emp.department);
    printf("Enter Salary: ");
    scanf("%f", &emp.salary);
    /* Insert into hash table */
    hashTable[index] = emp;
    occupied[index] = 1;
    printf("\nEmployee inserted successfully.\n");
    printf("Hash value = %d\n", startIndex);
    printf("Stored at index = %d\n", index);
}

/* Search employee using Employee ID */
void searchEmployee()
```

```

{
    int key;
    int index;
    int startIndex;
    printf("\nEnter Employee ID to search: ");
    scanf("%d", &key);
    index = hashFunction(key);
    startIndex = index;
    /* Linear probing search */
    while (occupied[index] == 1)
    {
        if (hashTable[index].empID == key)
        {
            printf("\nEmployee Record FOUND!\n");
            printf("-----\n");
            printf("Hash Table Index : %d\n", index);
            printf("Employee ID   : %d\n", hashTable[index].empID);
            printf("Name          : %s\n", hashTable[index].name);
            printf("Department     : %s\n", hashTable[index].department);
            printf("Salary         : %.2f\n", hashTable[index].salary);
            printf("Memory Address : %p\n",
                (void *)&hashTable[index]);
            printf("-----\n");
            return;
        }
        index = (index + 1) % tableSize;
        if (index == startIndex)
        {
            break;
        }
    }
    printf("\nEmployee ID %d NOT FOUND in the hash table.\n", key);
}

/* Display hash table */
void displayTable()
{
    int i;
    printf("\n===== \n");
    printf("          HASH TABLE CONTENTS\n");
    printf("===== \n");
    printf("%-8s %-12s %-20s %-15s %-12s\n",
        "Index", "Emp ID", "Name", "Department", "Salary");
    printf("-----\n");
    for (i = 0; i < tableSize; i++)
    {
        if (occupied[i] == 1)
        {
            printf("%-8d %-12d %-20s %-15s %-12.2f\n",
                i,

                hashTable[i].empID,
                hashTable[i].name,

```

```

        hashTable[i].department,
        hashTable[i].salary);
    printf(" Memory Address: %p\n",
        (void *)&hashTable[i]);
}
else
{
    printf("%-8d %-12s %-20s %-15s %-12s\n",
        i, "EMPTY", "-", "-", "-");
}
}

printf("===== \n");
}

/* Main function */
int main()
{
    int choice;
    printf("===== \n");
    printf("    EMPLOYEE RECORD MANAGEMENT SYSTEM\n");
    printf("    HASHING - LINEAR PROBING\n");
    printf("===== \n");
    printf("\nEnter hash table size (maximum %d): ", MAX);
    scanf("%d", &tableSize);
    if (tableSize <= 0 || tableSize > MAX)
    {
        printf("Invalid hash table size!\n");
        return 1;
    }
    initializeTable();
    do
    { printf("\n===== \n");
      printf("    HASH TABLE MENU\n");
      printf("===== \n");
      printf("1. Insert Employee Record\n");
      printf("2. Search Employee Record\n");
      printf("3. Display Hash Table\n");
      printf("4. Exit\n");
      printf("===== \n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
          case 1:
              insertEmployee();
              break;
          case 2:
              searchEmployee();

```

```
break;
case 3:
displayTable();
break;
case 4:
printf("\nExiting program...\n");
break;
default:
printf("\nInvalid choice! Please try again.\n");
}
}
while (choice != 4);
return 0;
}
```

*****OUTPUT*****

EMPLOYEE RECORD MANAGEMENT SYSTEM
HASHING - LINEAR PROBING

Enter hash table size (maximum 100): 10

Enter your choice: 1

Enter Employee ID: 101

Enter Employee Name: Rahul

Enter Department: CSE

Enter Salary: 45000

Employee inserted successfully.

Hash value = 1

Stored at index = 1

Enter your choice: 1

Enter Employee ID: 111

Enter Employee Name: Priya

Enter Department: ECE

Enter Salary: 50000

Employee inserted successfully.

Hash value = 1

Stored at index = 2

101 % 10 = 1

111 % 10 = 1

Enter your choice: 3

DATA STRUCTURES LABORATORY
HASH TABLE CONTENTS

1BCSL306

Index	Emp ID	Name	Department	Salary
0	EMPTY	-	-	-
1	101	Rahul	CSE	45000.00
		Memory Address: 0x55a1b2c3d...		
2	111	Priya	ECE	50000.00
		Memory Address: 0x55a1b2c4...		
3	EMPTY	-	-	-
4	EMPTY	-	-	-
5	EMPTY	-	-	-
6	EMPTY	-	-	-
7	EMPTY	-	-	-
8	EMPTY	-	-	-
9	EMPTY	-	-	-