

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ABHAY NY (1BM24CS400)

**in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
September 2024-January 2025**

**B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering**



This is to certify that the Lab work entitled “**DATA STRUCTURES**” carried out by ABHAY NY (**1BM24CS400**), who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - (**23CS3PCDST**)work prescribed for the said degree.

Prof. Sneha S Bagalkot

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda

Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1.	<p>Write a program to simulate the working of stack using an array with the following:</p> <p>a) Push</p> <p>b) Pop</p> <p>c) Display</p> <p>The program should print appropriate messages for stack overflow, stack underflow.</p>	6 – 12
2.	<p>WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)</p> <p>Program - Leetcode platform</p>	13 – 18
3.	<p>3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display</p> <p>The program should print appropriate messages for queue empty and queue overflow conditions</p> <p>3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display</p> <p>The program should print appropriate messages for queue empty and queue overflow conditions</p>	19 – 32
4.	<p>WAP to Implement Singly Linked List with following operations</p> <p>a) Create a linked list.</p> <p>b) Insertion of a node at first position, at any position and at end of list.</p>	33 – 41

	<p>Display the contents of the linked list.</p> <p>Program - Leetcode platform</p>	
5.	<p>WAP to Implement Singly Linked List with following operations</p> <p>a) Create a linked list.</p> <p>b) Deletion of first element, specified element and last element in the list.</p> <p>c) Display the contents of the linked list.</p> <p>Program - Leetcode platform</p>	42 – 51
6.	<p>6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.</p> <p>6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.</p>	52 – 66
7.	<p>WAP to Implement doubly link list with primitive operations</p> <p>a) Create a doubly linked list.</p> <p>b) Insert a new node to the left of the node.</p> <p>c) Delete the node based on a specific value</p> <p>d) Display the contents of the list</p> <p>Program - Leetcode platform</p>	67 – 73
8.	<p>Write a program</p> <p>a) To construct a binary Search tree.</p> <p>b) To traverse the tree using all the methods i.e., in-order, preorder and post order</p>	74 – 79

	c) To display the elements in the tree. Program - Leetcode platform	
9.	9a) Write a program to traverse a graph using BFS method. 9b) Write a program to check whether given graph is connected or not using DFS method.	80 – 85
10.	Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function $H: K \rightarrow L$ as $H(K) = K \bmod m$ (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.	86 – 88

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following:

a) Push

b) Pop

c) Display

The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 3

int top = -1;

int stack[SIZE];

void push(int item) {

    if (top == SIZE - 1) {

        printf("\nStack overflow");

    } else {

        top++;

        stack[top] = item;

        printf("\nElement %d pushed to stack", item);

    }

}

void pop() {

    if (top == -1) {

        printf("\nStack underflow");

    } else {
```

```

        printf("\nElement popped is %d", stack[top]);

        top--;

    }

}

void display() {

    if (top == -1) {

        printf("\nStack is empty");

    } else {

        printf("\nStack values:");

        for (int i = top; i >= 0; i--) {

            printf("\n%d", stack[i]);

        }

    }

}

int main() {

    int ch, item;

    for (;;) {

        printf("\n\n1: Push");

        printf("\n\n2: Pop");

        printf("\n\n3: Display");

        printf("\n\n4: Exit");

        printf("\nEnter your choice: ");

        scanf("%d", &ch);

```

```
switch (ch) {  
  
    case 1:  
  
        printf("Enter value to be pushed: ");  
  
        scanf("%d", &item);  
  
        push(item);  
  
        break;  
  
    case 2:  
  
        pop();  
  
        break;  
  
    case 3:  
  
        display();  
  
        break;  
  
    case 4:  
  
        exit(0);  
  
        break;  
  
    default:  
  
        printf("\nInvalid choice. Please try again.");  
  
        break;  
  
}  
  
}  
  
return 0;  
  
}
```


Output :

Pushing values:

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 1
Enter value to be pushed: 10

Element 10 pushed to stack
```

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 1
Enter value to be pushed: 20

Element 20 pushed to stack
```

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 1
Enter value to be pushed: 30

Element 30 pushed to stack
```

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 1
Enter value to be pushed: 40

Stack overflow
```

Poping values:

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 3
```

```
Stack values:
30
20
10
```

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 2

Element popped is 30
```

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 2

Element popped is 20
```

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 2

Element popped is 10
```

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 2

Stack underflow
```

Display:

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 3

Stack values:
30
20
10
```

```
1: Push
2: Pop
3: Display
4: Exit
Enter your choice: 3

Stack is empty
```

Lab program 2:

Program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).

Code:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 20

struct stack {

    int top;

    char data[SIZE];

};

typedef struct stack STACK;

void push(STACK *S, char item) {

    if (S->top < SIZE - 1) {

        S->data[++(S->top)] = item;

    } else {

        printf("Stack overflow!\n");

    }

}
```

```

char pop(STACK *S) {

    if (S->top != -1) {

        return S->data[(S->top)--];

    } else {

        printf("Stack underflow!\n");

        return '\0';

    }

}

```

```

int preced(char symbol) {

    switch (symbol) {

        case '^': return 5;

        case '*':

        case '/': return 3;

        case '+':

        case '-': return 1;

        default: return 0;

    }

}

```

```

void infixpostfix(char infix[20], STACK *S) {

    char postfix[20], symbol, temp;

    int i, j = 0;

```

```

for (i = 0; infix[i] != '\0'; i++) {

    symbol = infix[i];

    if (isalnum(symbol)) {

        postfix[j++] = symbol;

    } else {

        switch (symbol) {

            case '(':

                push(S, symbol);

                break;

            case ')':

                while (S->top != -1 && (temp = pop(S)) != '(') {

                    postfix[j++] = temp;

                }

                break;

            case '+':

            case '-':

            case '*':

            case '/':

            case '^':

                while (S->top != -1 && preced(S->data[S->top]) >= preced(symbol)) {

                    postfix[j++] = pop(S);

                }

```

```

        push(S, symbol);

        break;

    }

}

}

while (S->top != -1) {

    postfix[j++] = pop(S);

}

postfix[j] = '\0';

printf("\nPostfix expression is: %s\n", postfix);

}

int main() {

    char infix[20];

    STACK S;

    S.top = -1;

    printf("Enter infix expression: ");

    scanf("%s", infix);

    infixpostfix(infix, &S);

    printf("\nName: Abhay NY \nUSN: 24BECS404\n");

    return 0;

}

```


Output:

```
Enter infix expression: A+(B+C*D)^E  
Postfix expression is: ABCD*+E^+  
Name: Abhay NY  
USN: 24BECS404
```

Leet code:**3174. Clear Digits**

You are given a string *s*.

Your task is to remove all digits by doing this operation repeatedly:

Delete the first digit and the closest non-digit character to its left.

Return the resulting string after removing all digits.

Example 1:

Input: *s* = "abc"

Output: "abc"

Explanation:

There is no digit in the string.

Example 2:

Input: *s* = "cb34"

Output: ""

Explanation:

First, we apply the operation on *s*[2], and *s* becomes "c4".

Then we apply the operation on *s*[1], and *s* becomes "".

Constraints:

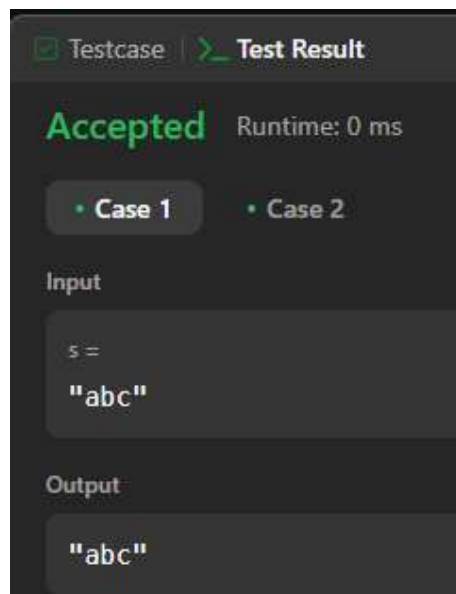
$1 \leq s.length \leq 100$

s consists only of lowercase English letters and digits.

The input is generated such that it is possible to delete all digits.

Code:

```
class Solution {
public:
    string clearDigits(string s) {
        int n = s.size();
        string ans = "";
        int i = 0;
        while(i < n)
        {
            if(isdigit(s[i]))
            {
                ans.pop_back();
            }
            else
            {
                ans.push_back(s[i]);
            }
            i++;
        }
        return ans;
    }
};
```



Lab program 3:

a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display

The program should print appropriate messages for queue empty and queue overflow conditions

Code:

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

int front = -1 , rear = -1 ;

int q[SIZE];

void enqueue (int item)
{
    if(rear==SIZE-1)

        printf("\n Queue is full ");

    else {

        rear=rear+1;

        q[rear]=item;

        if(front==-1)

            front=front+1;

    }
}
```

```

}

void dequeue()

{

    int del;

    if(front==-1)

        printf("\n Queue is empty");

    else{

        del=q[front];

        printf("\n Element deleted is : %d",del);

        if(front==rear)

        {

            front=-1;rear=-1;

        }

        else{

            front=front+1;

        }

    }

}

void display()

{

    int i;

    if(front==-1)

        printf("\n Queue is empty");

```

```

else{

    printf("\n Queue content \n");

    for (i=front;i<=rear;i++)

        printf("%d\t",q[i]);

    }

}

int main()

{

    int item,ch;

    for(;;)

    {

        printf("\n 1. Insert");

        printf("\n 2. Delete");

        printf("\n 3. Display");

        printf("\n 4. Exit");

        printf("\n read choice ");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1 : printf("\n Read element to be inserted ");

                    scanf("%d", & item);

                    enqueue(item);

```

```

        break;

    case 2 : dequeue();

        break;

    case 3 : display();

        break;

    default : exit (0);

}

}

return 0;

```

Output:

Enqueue and Display

```

PS C:\Users\Abhay> CD .\Desktop\
PS C:\Users\Abhay\Desktop> CD ".\C programs\"
PS C:\Users\Abhay\Desktop\C programs> gcc .\linear_queue_program_3.c
PS C:\Users\Abhay\Desktop\C programs> .\a.exe

1. Insert
2. Delete
3. Display
4. Exit
read choice 1

Read element to be inserted 10

1. Insert
2. Delete
3. Display
4. Exit
read choice 1

Read element to be inserted 20

1. Insert
2. Delete
3. Display
4. Exit
read choice 1

Read element to be inserted 30

1. Insert
2. Delete
3. Display
4. Exit
read choice 3

Queue content
10    20    30

```

Queue is full

```
PS C:\Users\Abhay\Desktop\C programs> .\a.
```

```
1. Insert  
2. Delete  
3. Display  
4. Exit  
read choice 1
```

```
Read element to be inserted 10
```

```
1. Insert  
2. Delete  
3. Display  
4. Exit  
read choice 1
```

```
Read element to be inserted 20
```

```
1. Insert  
2. Delete  
3. Display  
4. Exit  
read choice 1
```

```
Read element to be inserted 30
```

```
1. Insert  
2. Delete  
3. Display  
4. Exit  
read choice 1
```

```
Read element to be inserted 40
```

```
1. Insert  
2. Delete  
3. Display  
4. Exit  
read choice 1
```

```
Read element to be inserted 50
```

```
1. Insert  
2. Delete  
3. Display  
4. Exit  
read choice 1
```

```
Read element to be inserted 60
```

```
Queue is full
```

Deletion and printing the queue is empty

```
1. Insert
2. Delete
3. Display
4. Exit
read choice 2

Element deleted is : 10
1. Insert
2. Delete
3. Display
4. Exit
read choice 2

Element deleted is : 20
1. Insert
2. Delete
3. Display
4. Exit
read choice 3

Queue content
30
1. Insert
2. Delete
3. Display
4. Exit
read choice 2

Element deleted is : 30
1. Insert
2. Delete
3. Display
4. Exit
read choice 3

Queue is empty
```

Exit Command

```
1. Insert
2. Delete
3. Display
4. Exit
read choice 4
PS C:\Users\Abhay\Desktop\C programs> █
```


b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display

The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>

#include <stdlib.h>

#define size 5

int f=-1;

int r=-1;

int q[size];

void enqueue(int item)

{

    if(f==(r+1)%size)

        printf("Queue is full");

    else{

        r=(r+1)%size;

        q[r]=item;

        if(f==-1)

            f=f+1;

    }

}

void dequeue()

{
```

```

    if(f== -1)

    printf("Queue is empty");

    else{

        printf("\n Elements deleted is %d ",q[f]);

        if(f==r)

        {

            f=-1;

            r=-1;

        }

        else{

            f=(f+1)%size;

        }

    }

}

void display()

{

    int i;

    if(f== -1)

        printf("Queue is empty");

    else{

        printf("\n Content of queue : \n");

        for(i=f ; i!=r ; i=(i+1)%size)

            printf("%d \t",q[i]);

    }

}

```

```

        printf("%d \t", q[r]);

    }

}

int main()

{

    int ch , item;

    for(;;)

    {

        printf("\n 1. Insert");

        printf("\n 2. Delete");

        printf("\n 3. Display");

        printf("\n 4. Exit");

        printf("\n Read choice : ");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1 : printf("\n Enter element to be inserted : ");

                    scanf("%d",&item);

                    enqueue(item);

                    break;

            case 2 : dequeue();

                    break;

            case 3 : display();

```

```
        break;

        default : exit(0);

    }

    }return 0;}
```

Enqueue into the circular queue and displaying

```
PS C:\Users\Abhay\Desktop\C programs> .\a.exe

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 10

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 20

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 30

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 40

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 3

Content of queue :
10      20      30      40
```

Circular queue overflow

```
Content of queue :  
10      20      30      40  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Read choice : 1  
  
Enter element to be inserted : 50  
  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Read choice : 1  
  
Enter element to be inserted : 60  
Queue is full
```

Deleting elements form the queue till queue is empty

```
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 2

Elements deleted is 10
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 2

Elements deleted is 20
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 2

Elements deleted is 30
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 2

Elements deleted is 40
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 2

Elements deleted is 50
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 3
Queue is empty
```

Checking the Circular list working

First we enter 3 elements and then delete once and enqueue 40 to the queue .

```
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 10

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 20

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 30

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 3

Content of queue :
10      20      30
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 2

Elements deleted is 10
```

```
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 3

Content of queue :
20      30
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 40

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 3

Content of queue :
20      30      40
```


Lab program 4:**WAP to Implement Singly Linked List with following operations**

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node* next;

};

void createLinkedList(struct Node** head);

void insertAtBeginning(struct Node** head, int data);

void insertAtPosition(struct Node** head, int data, int position);

void insertAtEnd(struct Node** head, int data);

void displayList(struct Node* head);

int main() {

    struct Node* head = NULL; // Initialize the linked list as empty

    int choice, data, position;

    while (1) {

        printf("\nMenu:\n");

        printf("1. Create Linked List\n");

        printf("2. Insert at Beginning\n");
```

```
printf("3. Insert at Position\n");

printf("4. Insert at End\n");

printf("5. Display List\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

    case 1:

        createLinkedList(&head);

        break;

    case 2:

        printf("Enter data to insert: ");

        scanf("%d", &data);

        insertAtBeginning(&head, data);

        break;

    case 3:

        printf("Enter data to insert: ");

        scanf("%d", &data);

        printf("Enter position: ");

        scanf("%d", &position);

        insertAtPosition(&head, data, position);

        break;

    case 4:
```

```

        printf("Enter data to insert: ");

        scanf("%d", &data);

        insertAtEnd(&head, data);

        break;

    case 5:

        displayList(head);

        break;

    case 6:

        printf("Exiting...\n");

        exit(0);

    default:

        printf("Invalid choice! Please try again.\n");

    }

}

return 0;

}

void createLinkedList(struct Node** head) {

    int n, data;

    printf("Enter the number of nodes: ");

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {

        printf("Enter data for node %d: ", i + 1);

        scanf("%d", &data);

```

```

        insertAtEnd(head, data);

    }

}

void insertAtBeginning(struct Node** head, int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    if (!newNode) {

        printf("Memory allocation failed!\n");

        return;

    }

    newNode->data = data;

    newNode->next = *head;

    *head = newNode;

}

void insertAtPosition(struct Node** head, int data, int position) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    if (!newNode) {

        printf("Memory allocation failed!\n");

        return;

    }

    newNode->data = data;

    if (position == 1) {

        newNode->next = *head;

        *head = newNode;

    }

}

```

```

        return;

    }

    struct Node* temp = *head;

    for (int i = 1; i < position - 1 && temp != NULL; i++) {

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Position out of bounds!\n");

        free(newNode);

        return;

    }

    newNode->next = temp->next;

    temp->next = newNode;

}

void insertAtEnd(struct Node** head, int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    if (!newNode) {

        printf("Memory allocation failed!\n");

        return;

    }

    newNode->data = data;

    newNode->next = NULL;

```

```

    if (*head == NULL) {

        *head = newNode;

        return;

    }

    struct Node* temp = *head;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = newNode;

}

void displayList(struct Node* head) {

    if (head == NULL) {

        printf("The list is empty!\n");

        return;

    }

    struct Node* temp = head;

    printf("Linked List: ");

    while (temp != NULL) {

        printf("%d -> ", temp->data);

        temp = temp->next;

    }

    printf("NULL\n");

}

```

Output:

Menu:

1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exit

Enter your choice: 1

Enter the number of nodes: 5

Enter data for node 1: 10

Enter data for node 2: 20

Enter data for node 3: 30

Enter data for node 4: 40

Enter data for node 5: 50

Menu:

1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exit

Enter your choice: 2

Enter data to insert: 60

Menu:

1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exit

Enter your choice: 4

Enter data to insert: 70

Menu:

1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exit

Enter your choice: 5

Linked List: 60 -> 10 -> 20 -> 30 -> 40 -> 50 -> 70 -> NULL

Leet code Questions

2073. Time needed to buy tickets

There are n people in a line queuing to buy tickets, where the 0th person is at the front of the line and the $(n - 1)$ th person is at the back of the line.

You are given a 0-indexed integer array `tickets` of length n where the number of tickets that the i th person would like to buy is `tickets[i]`.

Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to go back to the end of the line (which happens instantaneously) in order to buy more tickets. If a person does not have any tickets left to buy, the person will leave the line.

Return the time taken for the person initially at position k (0-indexed) to finish buying tickets.

```
class Solution {
public:
    int timeRequiredToBuy(vector<int>& tickets, int k) {
        int n = tickets.size();
        int time = 0;


        // If person k only needs one ticket, return the time to buy it
        if (tickets[k] == 1)
            return k + 1;

        // Continue buying tickets until person k buys all their tickets
        while (tickets[k] > 0) {
            for (int i = 0; i < n; i++) {
                // Buy a ticket at index 'i' if available
                if (tickets[i] != 0) {
                    tickets[i]--;
                    time++;
                }
            }

            // If person k bought all their tickets, return the time
            if (tickets[k] == 0)
                return time;
        }

        return time;
    }
};
```


Output:

☒ Testcase |  Test Result

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

tickets =
[2,3,2]

k =
2

Output

6

Expected

6

Lab program 5:**WAP to Implement Singly Linked List with following operations**

a) Create a linked list.

b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

Code:

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node* next;

};

void createLinkedList(struct Node** head);

void deleteFirst(struct Node** head);

void deleteSpecified(struct Node** head, int key);

void deleteLast(struct Node** head);

void displayList(struct Node* head);

int main() {

    struct Node* head = NULL; // Initialize the linked list as empty

    int choice, data;

    while (1) {

        printf("\nMenu:\n");

        printf("1. Create Linked List\n");
```

```
printf("2. Delete First Element\n");

printf("3. Delete Specified Element\n");

printf("4. Delete Last Element\n");

printf("5. Display List\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

    case 1:

        createLinkedList(&head);

        break;

    case 2:

        deleteFirst(&head);

        break;

    case 3:

        printf("Enter the element to delete: ");

        scanf("%d", &data);

        deleteSpecified(&head, data);

        break;

    case 4:

        deleteLast(&head);

        break;

    case 5:
```

```

        displayList(head);

        break;

    case 6:

        printf("Exiting...\n");

        exit(0);

    default:

        printf("Invalid choice! Please try again.\n");

    }

}

return 0;

}

void createLinkedList(struct Node** head) {

    int n, data;

    printf("Enter the number of nodes: ");

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {

        printf("Enter data for node %d: ", i + 1);

        scanf("%d", &data);

        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

        if (!newNode) {

            printf("Memory allocation failed!\n");

            return;

        }
    }
}

```

```

newNode->data = data;

newNode->next = NULL;

if (*head == NULL) {

    *head = newNode;

} else {

    struct Node* temp = *head;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = newNode;

}

}

}

void deleteFirst(struct Node** head) {

    if (*head == NULL) {

        printf("The list is empty!\n");

        return;

    }

    struct Node* temp = *head;

    *head = (*head)->next;

    free(temp);

    printf("First element deleted.\n");

}

```

```

void deleteSpecified(struct Node** head, int key) {

    if (*head == NULL) {

        printf("The list is empty!\n");

        return;

    }

    struct Node* temp = *head;

    struct Node* prev = NULL;

    if (temp != NULL && temp->data == key) {

        *head = temp->next;

        free(temp);

        printf("Element %d deleted.\n", key);

        return;

    }

    while (temp != NULL && temp->data != key) {

        prev = temp;

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Element %d not found in the list!\n", key);

        return;

    }

    prev->next = temp->next;

```

```

        free(temp);

        printf("Element %d deleted.\n", key);
    }

void deleteLast(struct Node** head) {

    if (*head == NULL) {

        printf("The list is empty!\n");

        return;

    }

    struct Node* temp = *head;

    struct Node* prev = NULL;

    if (temp->next == NULL) {

        *head = NULL;

        free(temp);

        printf("Last element deleted.\n");

        return;

    }

    while (temp->next != NULL) {

        prev = temp;

        temp = temp->next;

    }

    prev->next = NULL;

    free(temp);

    printf("Last element deleted.\n");
}

```

```
}  
  
void displayList(struct Node* head) {  
  
    if (head == NULL) {  
  
        printf("The list is empty!\n");  
  
        return;  
  
    }  
  
    struct Node* temp = head;  
  
    printf("Linked List: ");  
  
    while (temp != NULL) {  
  
        printf("%d -> ", temp->data);  
  
        temp = temp->next;  
  
    }  
  
    printf("NULL\n");  
  
}
```


Output:

Menu:

1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display List
6. Exit

Enter your choice: 1

Enter the number of nodes: 5

Enter data for node 1: 10

Enter data for node 2: 20

Enter data for node 3: 30

Enter data for node 4: 40

Enter data for node 5: 50

Menu:

1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display List
6. Exit

Enter your choice: 2

First element deleted.

Menu:

1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display List
6. Exit

Enter your choice: 3

Enter the element to delete: 30

Element 30 deleted.

Menu:

1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display List
6. Exit

Enter your choice: 4

Last element deleted.

Menu:

1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display List
6. Exit

Enter your choice: 5

Linked List: 20 -> 40 -> NULL

1823. Find the winner of the circular game

There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in clockwise order. More formally, moving clockwise from the i th friend brings you to the $(i+1)$ th friend for $1 \leq i < n$, and moving clockwise from the n th friend brings you to the 1st friend.

The rules of the game are as follows:

Start at the 1st friend.

Count the next k friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once.

The last friend you counted leaves the circle and loses the game.

If there is still more than one friend in the circle, go back to step 2 starting from the friend immediately clockwise of the friend who just lost and repeat.

Else, the last friend in the circle wins the game.

Given the number of friends, n , and an integer k , return the winner of the game.

```
class Solution {
public:
    int findTheWinner(int n, int k) {
        // Initialize vector of N friends, labeled from 1-N
        vector<int> circle;
        for (int i = 1; i <= n; i++) {
            circle.push_back(i);
        }

        // Maintain the index of the friend to start the count on
        int startIndex = 0;

        // Perform eliminations while there is more than 1 friend left
        while (circle.size() > 1) {
            // Calculate the index of the friend to be removed
            int removalIndex = (startIndex + k - 1) % circle.size();

            // Erase the friend at removalIndex
            circle.erase(circle.begin() + removalIndex);

            // Update startIndex for the next round
            startIndex = removalIndex;
        }
    }
};
```

```
        return circle.front();  
    }  
};
```

Output:

The screenshot shows a 'Test Result' window with a dark theme. At the top, it says 'Accepted' in green and 'Runtime: 0 ms'. Below this, there are two tabs: 'Case 1' (selected) and 'Case 2'. Under 'Case 1', the 'Input' section shows 'n =' with the value '5' and 'k =' with the value '2'. The 'Output' section shows the value '3'. The 'Expected' section also shows the value '3'. A small copy icon is visible next to the 'k =' input field.

Section	Value
Input (n)	5
Input (k)	2
Output	3
Expected	3

Lab program 6:

a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node* next;

};

struct Node* createNode(int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

void append(struct Node** head, int data) {

    struct Node* newNode = createNode(data);

    if (*head == NULL) {

        *head = newNode;

        return;

    }

    struct Node* temp = *head;

    while (temp->next != NULL) {
```

```

        temp = temp->next;

    }

    temp->next = newNode;
}

void display(struct Node* head) {

    while (head != NULL) {

        printf("%d -> ", head->data);

        head = head->next;

    }

    printf("NULL\n");

}

void sortList(struct Node** head) {

    if (*head == NULL || (*head)->next == NULL) {

        return;

    }

    struct Node *i, *j;

    int temp;

    for (i = *head; i != NULL; i = i->next) {

        for (j = i->next; j != NULL; j = j->next) {

            if (i->data > j->data) {

                temp = i->data;

                i->data = j->data;

                j->data = temp; } } } }

```

```

void reverseList(struct Node** head) {

    struct Node *prev = NULL, *current = *head, *next = NULL;

    while (current != NULL) {

        next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    *head = prev;

}

void concatenateLists(struct Node** head1, struct Node** head2) {

    if (*head1 == NULL) {

        *head1 = *head2;

        return;

    }

    struct Node* temp = *head1;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = *head2;

}

int main() {

    struct Node* list1 = NULL;

```

```

struct Node* list2 = NULL;

int choice, data;

do {

    printf("\nMenu:\n");

    printf("1. Append to List 1\n");

    printf("2. Display List 1\n");

    printf("3. Sort List 1\n");

    printf("4. Reverse List 1\n");

    printf("5. Append to List 2\n");

    printf("6. Display List 2\n");

    printf("7. Concatenate List 2 to List 1\n");

    printf("8. Exit\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);

    switch (choice) {

        case 1:

            printf("Enter a number to append to List 1: ");

            scanf("%d", &data);

            append(&list1, data);

            break;

        case 2:

            printf("List 1: ");

            display(list1);

```

```
        break;

    case 3:

        sortList(&list1);

        printf("Sorted List 1: ");

        display(list1);

        break;

    case 4:

        reverseList(&list1);

        printf("Reversed List 1: ");

        display(list1);

        break;

    case 5:

        printf("Enter a number to append to List 2: ");

        scanf("%d", &data);

        append(&list2, data);

        break;

    case 6:

        printf("List 2: ");

        display(list2);

        break;

    case 7:

        concatenateLists(&list1, &list2);

        printf("Concatenated List 1: ");
```



```
        display(list1);

        break;

    case 8:

        printf("Exiting program.\n");

        break;

    default:

        printf("Invalid choice. Please try again.\n");

    }

} while (choice != 8);

return 0;

}
```

```
Menu:
1. Append to List 1
2. Display List 1
3. Sort List 1
4. Reverse List 1
5. Append to List 2
6. Display List 2
7. Concatenate List 2 to List 1
8. Exit
Enter your choice: 2
List 1: 10 -> 20 -> 30 -> 40 -> 50 -> NULL

Menu:
1. Append to List 1
2. Display List 1
3. Sort List 1
4. Reverse List 1
5. Append to List 2
6. Display List 2
7. Concatenate List 2 to List 1
8. Exit
Enter your choice: 3
Sorted List 1: 10 -> 20 -> 30 -> 40 -> 50 -> NULL

Menu:
1. Append to List 1
2. Display List 1
3. Sort List 1
4. Reverse List 1
5. Append to List 2
6. Display List 2
7. Concatenate List 2 to List 1
8. Exit
Enter your choice: 4
Reversed List 1: 50 -> 40 -> 30 -> 20 -> 10 -> NULL

Menu:
1. Append to List 1
2. Display List 1
3. Sort List 1
4. Reverse List 1
5. Append to List 2
6. Display List 2
7. Concatenate List 2 to List 1
8. Exit
Enter your choice: 5
Enter a number to append to List 2: 60

Menu:
1. Append to List 1
2. Display List 1
3. Sort List 1
4. Reverse List 1
5. Append to List 2
6. Display List 2
7. Concatenate List 2 to List 1
8. Exit
Enter your choice: 7
Concatenated List 1: 50 -> 40 -> 30 -> 20 -> 10 -> 60 -> NULL
```

b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node* next;

};

struct Node* createNode(int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    if (!newNode) {

        printf("Memory allocation error\n");

        exit(1);

    }

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

void push(struct Node** top, int data) {

    struct Node* newNode = createNode(data);

    newNode->next = *top;

    *top = newNode;

    printf("Pushed %d to stack\n", data);

}
```

```

int pop(struct Node** top) {

    if (*top == NULL) {

        printf("Stack underflow\n");

        return -1;

    }

    struct Node* temp = *top;

    int popped = temp->data;

    *top = (*top)->next;

    free(temp);

    return popped;

}

void enqueue(struct Node** front, struct Node** rear, int data) {

    struct Node* newNode = createNode(data);

    if (*rear == NULL) {

        *front = *rear = newNode;

        printf("Enqueued %d to queue\n", data);

        return;

    }

    (*rear)->next = newNode;

    *rear = newNode;

    printf("Enqueued %d to queue\n", data);

}

int dequeue(struct Node** front, struct Node** rear) {

```

```

    if (*front == NULL) {

        printf("Queue underflow\n");

        return -1;

    }

    struct Node* temp = *front;

    int dequeued = temp->data;

    *front = (*front)->next;

    if (*front == NULL)

        *rear = NULL;

    free(temp);

    return dequeued;

}

void display(struct Node* node) {

    if (node == NULL) {

        printf("List is empty\n");

        return;

    }

    while (node != NULL) {

        printf("%d -> ", node->data);

        node = node->next;

    }

    printf("NULL\n");

}

```

```

int main() {

    struct Node* stackTop = NULL;

    struct Node* queueFront = NULL;

    struct Node* queueRear = NULL;

    int choice, value;

    do {

        printf("\nMenu:\n");

        printf("1. Push to Stack\n");

        printf("2. Pop from Stack\n");

        printf("3. Display Stack\n");

        printf("4. Enqueue to Queue\n");

        printf("5. Dequeue from Queue\n");

        printf("6. Display Queue\n");

        printf("0. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to push: ");

                scanf("%d", &value);

                push(&stackTop, value);

                break;

            case 2:

```

```
        value = pop(&stackTop);

        if (value != -1)

            printf("Popped %d from stack\n", value);

        break;

case 3:

    printf("Stack contents: ");

    display(stackTop);

    break;

case 4:

    printf("Enter value to enqueue: ");

    scanf("%d", &value);

    enqueue(&queueFront, &queueRear, value);

    break;

case 5:

    value = dequeue(&queueFront, &queueRear);

    if (value != -1)

        printf("Dequeued %d from queue\n", value);

    break;

case 6:

    printf("Queue contents: ");

    display(queueFront);

    break;

case 0:
```

```
        printf("Exiting program...\n");

        break;

default:

        printf("Invalid choice! Please try again.\n");

    }

} while (choice != 0);

return 0;

}
```


Stack:

```
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10 to stack
```

```
Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 1
Enter value to push: 20
Pushed 20 to stack
```

```
Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 1
Enter value to push: 30
Pushed 30 to stack
```

```
Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 1
Enter value to push: 40
Pushed 40 to stack
```

```
Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 2
Popped 40 from stack
```

```
Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 3
Stack contents: 30 -> 20 -> 10 -> NULL
```

Queue:

```
Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 4
Enter value to enqueue: 10
Enqueued 10 to queue

Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 4
Enter value to enqueue: 20
Enqueued 20 to queue

Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 4
Enter value to enqueue: 30
Enqueued 30 to queue

Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 4
Enter value to enqueue: 40
Enqueued 40 to queue

Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 5
Dequeued 10 from queue

Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
0. Exit
Enter your choice: 6
Queue contents: 20 -> 30 -> 40 -> NULL
```

Lab program 7:

WAP to Implement doubly link list with primitive operations

- a) Create a doubly linked list.**
- b) Insert a new node to the left of the node.**
- c) Delete the node based on a specific value**
- d) Display the contents of the list**

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

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

```
typedef struct Node {
```

```
    int data;
```

```
    struct Node* prev;
```

```
    struct Node* next;
```

```
} Node;
```

```
Node* createNode(int data) {
```

```
    Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    newNode->data = data;
```

```
    newNode->prev = NULL;
```

```
    newNode->next = NULL;
```

```
    return newNode;
```

```
}
```

```
void append(Node** head, int data) {
```

```
    Node* newNode = createNode(data);
```

```
    if (*head == NULL) {
```

```

        *head = newNode;

        return;
    }

    Node* temp = *head;

    while (temp->next) {

        temp = temp->next;

    }

    temp->next = newNode;

    newNode->prev = temp;

}

void insertLeft(Node** head, int target, int data) {

    Node* newNode = createNode(data);

    Node* current = *head;

    while (current && current->data != target) {

        current = current->next;

    }

    if (!current) {

        printf("Node with value %d not found.\n", target);

        free(newNode);

        return;

    }

    newNode->next = current;

    newNode->prev = current->prev;

```

```

    if (current->prev) {

        current->prev->next = newNode;

    } else {

        *head = newNode;

    }

    current->prev = newNode;
}

void deleteNode(Node** head, int value) {

    Node* current = *head;

    while (current && current->data != value) {

        current = current->next;

    }

    if (!current) {

        printf("Node with value %d not found.\n", value);

        return;

    }

    if (current->prev) {

        current->prev->next = current->next;

    } else {

        *head = current->next;

    }

    if (current->next) {

        current->next->prev = current->prev;

```

```

    }

    free(current);

    printf("Node with value %d deleted.\n", value);
}

void displayList(Node* head) {

    Node* current = head;

    if (!head) {

        printf("The list is empty.\n");

        return;

    }

    printf("List contents: ");

    while (current) {

        printf("%d ", current->data);

        current = current->next;

    }

    printf("\n");

}

int main() {

    Node* head = NULL;

    int choice, data, target;

    while (1) {

        printf("\nDoubly Linked List Operations:\n");

        printf("1. Create/Add a Node\n");

```

```
printf("2. Insert a Node to the Left of a Specific Node\n");

printf("3. Delete a Node by Value\n");

printf("4. Display the List\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

    case 1:

        printf("Enter the value to add: ");

        scanf("%d", &data);

        append(&head, data);

        break;

    case 2:

        printf("Enter the value to insert: ");

        scanf("%d", &data);

        printf("Enter the target value to insert to the left of: ");

        scanf("%d", &target);

        insertLeft(&head, target, data);

        break;

    case 3:

        printf("Enter the value to delete: ");

        scanf("%d", &data);

        deleteNode(&head, data);
```

```
        break;

    case 4:

        displayList(head);

        break;

    case 5:

        printf("Exiting...\n");

        return 0;

    default:

        printf("Invalid choice! Please try again.\n");

    }

}

}
```


Output:

Doubly Linked List Operations:

1. Create/Add a Node
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display the List
5. Exit

Enter your choice: 1

Enter the value to add: 10

Doubly Linked List Operations:

1. Create/Add a Node
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display the List
5. Exit

Enter your choice: 1

Enter the value to add: 20

Doubly Linked List Operations:

1. Create/Add a Node
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display the List
5. Exit

Enter your choice: 1

Enter the value to add: 30

Doubly Linked List Operations:

1. Create/Add a Node
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display the List
5. Exit

Enter your choice: 1

Enter the value to add: 40

Doubly Linked List Operations:

1. Create/Add a Node
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display the List
5. Exit

Enter your choice: 2

Enter the value to insert: 50

Enter the target value to insert to the left of: 20

Doubly Linked List Operations:

1. Create/Add a Node
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display the List
5. Exit

Enter your choice: 3

Enter the value to delete: 10

Node with value 10 deleted.

Doubly Linked List Operations:

1. Create/Add a Node
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display the List
5. Exit

Enter your choice: 4

List contents: 50 20 30 40

Lab program 8:

Write a program

a) To construct a binary Search tree.

**b) To traverse the tree using all the methods i.e., in-order,
preorder and post order**

c) To display the elements in the tree.

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

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

```
typedef struct Node {
```

```
    int data;
```

```
    struct Node *left, *right;
```

```
} Node;
```

```
Node* createNode(int value) {
```

```
    Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    newNode->data = value;
```

```
    newNode->left = newNode->right = NULL;
```

```
    return newNode;
```

```
}
```

```
Node* insert(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);

    }

    return root;

}

void inorderTraversal(Node* root) {

    if (root != NULL) {

        inorderTraversal(root->left);

        printf("%d ", root->data);

        inorderTraversal(root->right);

    }

}

void preorderTraversal(Node* root) {

    if (root != NULL) {

        printf("%d ", root->data);

        preorderTraversal(root->left);

        preorderTraversal(root->right);

    }

}

void postorderTraversal(Node* root) {

    if (root != NULL) {

        postorderTraversal(root->left);

```

```

        postorderTraversal(root->right);

        printf("%d ", root->data);

    }

}

int main() {

    Node* root = NULL;

    int choice, value;

    do {

        printf("\nBinary Search Tree Menu\n");

        printf("1. Insert an element\n");

        printf("2. Display In-order traversal\n");

        printf("3. Display Pre-order traversal\n");

        printf("4. Display Post-order traversal\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter the value to insert: ");

                scanf("%d", &value);

                root = insert(root, value);

                printf("Value inserted successfully.\n");

                break;

```

```
case 2:

    printf("In-order Traversal: ");

    inorderTraversal(root);

    printf("\n");

    break;

case 3:

    printf("Pre-order Traversal: ");

    preorderTraversal(root);

    printf("\n");

    break;

case 4:

    printf("Post-order Traversal: ");

    postorderTraversal(root);

    printf("\n");

    break;

case 5:

    printf("Exiting...\n");

    break;

default:

    printf("Invalid choice! Please try again.\n");

}

} while (choice != 5);
```

```
    return 0;
}
```

```
Binary Search Tree Menu
1. Insert an element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 10
Value inserted successfully.
```

```
Binary Search Tree Menu
1. Insert an element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 30
Value inserted successfully.
```

```
Binary Search Tree Menu
1. Insert an element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 20
Value inserted successfully.
```

```
Binary Search Tree Menu
1. Insert an element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 60
Value inserted successfully.
```

```
Binary Search Tree Menu
1. Insert an element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 70
Value inserted successfully.
```

Binary Search Tree Menu

1. Insert an element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit

Enter your choice: 2

In-order Traversal: 10 20 30 60 70

Binary Search Tree Menu

1. Insert an element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit

Enter your choice: 3

Pre-order Traversal: 10 30 20 60 70

Binary Search Tree Menu

1. Insert an element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit

Enter your choice: 4

Post-order Traversal: 20 70 60 30 10

Lab program 9:**A) Write a program to traverse a graph using BFS method.**

```
#include <stdio.h>

#include <stdlib.h>

int adj[MAX][MAX];

int visited[MAX];

int queue[MAX];

int front = -1, rear = -1;

void enqueue(int vertex) {

    if (rear == MAX - 1) {

        printf("Queue overflow\n");

        return;

    }

    if (front == -1)

        front = 0;

    queue[++rear] = vertex;

}

int dequeue() {

    if (front == -1 || front > rear) {

        printf("Queue underflow\n");

        return -1;

    }

    return queue[front++];

}
```



```

}

int isEmpty() {

    return (front == -1 || front > rear);

}

void bfs(int startVertex, int n) {

    enqueue(startVertex);

    visited[startVertex] = 1;

    printf("BFS Traversal starting from vertex %d:\n", startVertex);

    while (!isEmpty()) {

        int currentVertex = dequeue();

        printf("%d ", currentVertex);

        for (int i = 0; i < n; i++) {

            if (adj[currentVertex][i] == 1 && !visited[i]) {

                enqueue(i);

                visited[i] = 1;

            }

        }

    }

    printf("\n");

}

int main() {

    int n, edges, v1, v2, startVertex;

```

```

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter the number of edges: ");

scanf("%d", &edges);

for (int i = 0; i < n; i++) {

    for (int j = 0; j < n; j++) {

        adj[i][j] = 0;

    }

    visited[i] = 0;

}

printf("Enter the edges (vertex1 vertex2):\n");

for (int i = 0; i < edges; i++) {

    scanf("%d %d", &v1, &v2);

    adj[v1][v2] = 1;

    adj[v2][v1] = 1; // For undirected graph

}

printf("Enter the starting vertex for BFS: ");

scanf("%d", &startVertex);

// Perform BFS

bfs(startVertex, n);

return 0;

}

```

Output:

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges (vertex1 vertex2):
0 1
0 2
1 3
1 4
2 4
3 4
Enter the starting vertex for BFS: 0
BFS Traversal starting from vertex 0:
0 1 2 3 4
```

B) Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int adj[MAX][MAX];

int visited[MAX];

void dfs(int vertex, int n) {

    visited[vertex] = 1;

    printf("%d ", vertex);

    for (int i = 0; i < n; i++) {

        if (adj[vertex][i] == 1 && !visited[i]) {

            dfs(i, n);

        }

    }

}

int isConnected(int n) {
```

```

    for (int i = 0; i < n; i++) {

        visited[i] = 0;

    }

    dfs(0, n);

    for (int i = 0; i < n; i++) {

        if (visited[i] == 0) {

            return 0;

        }

    }

    return 1;

}

int main() {

    int n, edges, v1, v2;

    printf("Enter the number of vertices: ");

    scanf("%d", &n);

    printf("Enter the number of edges: ");

    scanf("%d", &edges);

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            adj[i][j] = 0; // No edge by default

        }

    }

    printf("Enter the edges (vertex1 vertex2):\n");

```

```

for (int i = 0; i < edges; i++) {

    scanf("%d %d", &v1, &v2);

    adj[v1][v2] = 1;

    adj[v2][v1] = 1;

}

if (isConnected(n)) {

    printf("The graph is connected.\n");

} else {

    printf("The graph is not connected.\n");

}

return 0;

}

Enter the number of vertices: 5
Enter the number of edges: 4
Enter the edges (vertex1 vertex2):
0 1
1 2
2 3
3 4
0 1 2 3 4 The graph is connected.


Enter the number of vertices: 5
Enter the number of edges: 2
Enter the edges (vertex1 vertex2):
0 1
1 2
0 1 2 The graph is not connected.

```

Lab program 10:

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.

Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.

Let the keys in K and addresses in L are integers.

Design and develop a Program in C that uses Hash function $H: K \rightarrow L$ as $H(K)=K \bmod m$ (remainder method), and implement hashing technique to map a given key K to the address space L.

Resolve the collision (if any) using linear probing.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_KEYS 100

void initializeHashTable(int hashTable[], int m) {
    for (int i = 0; i < m; i++) {
        hashTable[i] = -1;
    }
}

int hashFunction(int K, int m) {
    return K % m;
}

void insertIntoHashTable(int hashTable[], int m, int key) {
    int index = hashFunction(key, m);
```

```

while (hashTable[index] != -1) {

    index = (index + 1) % m;

}

hashTable[index] = key;

printf("Key %d inserted at index %d\n", key, index);

}

void displayHashTable(int hashTable[], int m) {

    printf("\nHash Table:\n");

    for (int i = 0; i < m; i++) {

        if (hashTable[i] != -1) {

            printf("Index %d -> Key %d\n", i, hashTable[i]);

        } else {

            printf("Index %d -> Empty\n", i);

        }

    }

}

int main() {

    int m, n;

    int keys[MAX_KEYS]; // Array to store the keys (employee records)

    printf("Enter the number of memory locations in hash table (m): ");

    scanf("%d", &m);

    printf("Enter the number of keys (n): ");

    scanf("%d", &n);

```

```

printf("Enter the keys (4-digit integers):\n");

for (int i = 0; i < n; i++) {

    scanf("%d", &keys[i]);

}

int hashTable[m];

initializeHashTable(hashTable, m);

for (int i = 0; i < n; i++) {

    insertIntoHashTable(hashTable, m, keys[i]);

}

displayHashTable(hashTable, m);

return 0;

}

```

Enter the number of memory locations in hash table (m): 10
Enter the number of keys (n): 5
Enter the keys (4-digit integers):
1234 2345 3456 4567 5678
Key 1234 inserted at index 4
Key 2345 inserted at index 5
Key 3456 inserted at index 6
Key 4567 inserted at index 7
Key 5678 inserted at index 8

Hash Table:
Index 0 -> Empty
Index 1 -> Empty
Index 2 -> Empty
Index 3 -> Empty
Index 4 -> Key 1234
Index 5 -> Key 2345
Index 6 -> Key 3456
Index 7 -> Key 4567
Index 8 -> Key 5678
Index 9 -> Empty