VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

MAKADIA RISHIT DILIPBHAI

(1BM23CS177)

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 **MAKADIA RISHIT DILIPBHAI (1BM23CS177)**, 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. Namratha M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

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.

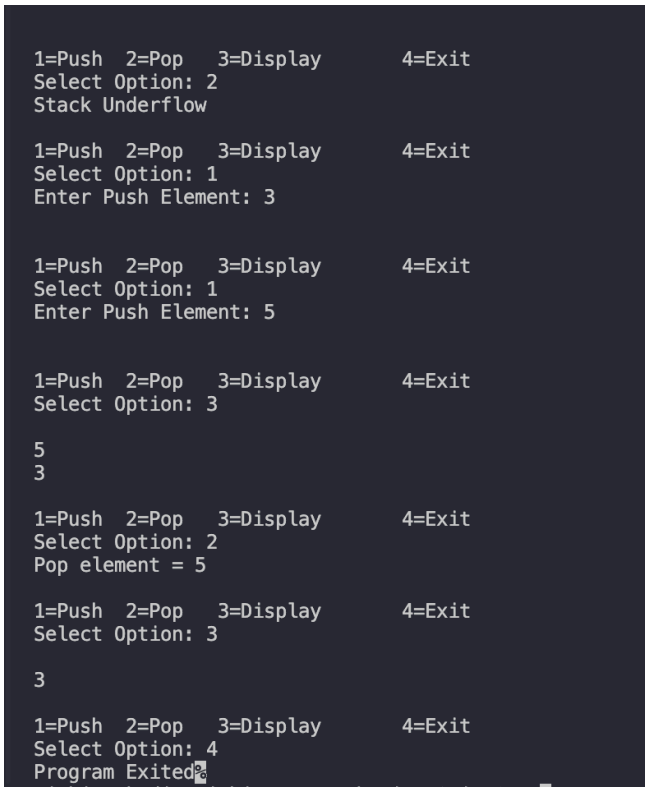
Lab program 1:

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>

#define MAX 5
int stack[MAX];
int top = -1;

void push(int ele)
{
   if (top==(MAX-1))
      printf("Stack Overflow");
   else
   {
      top = top + 1;
      stack[top] = ele;
   }
}

void pop()
{
   if (top==-1)
      printf("Stack Underflow ");
   else
   {
      int popele = stack[top];
      printf("Pop element = %d", popele);
      top--;
   }
}

void display()
{
   if (top==-1)
      printf("Stack is empty");
   else
   {
      for (int i = top; i >= 0; i--)
      {
         printf("\n%d", stack[i]);
```

```
        }
    }
}

int main()
{
    int opt, in;
    while (1)
    {
        printf("\n\n1=Push\t2=Pop\t3=Display\t4=Exit");
        printf("\nSelect Option: ");
        scanf("%d", &opt);

        switch(opt)
        {
            case 1:
            {
                printf("Enter Push Element: ");
                scanf("%d", &in);
                push(in);
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                printf("Program Exited");
                return 0;
            }
            default:
                printf("Invalid Operations");
        }
    }
    return 0;
}
```

Output:


1=Push 2=Pop 3=Display 4=Exit
Select Option: 2
Stack Underflow
1=Push 2=Pop 3=Display 4=Exit
Select Option: 1
Enter Push Element: 3
1=Push 2=Pop 3=Display 4=Exit
Select Option: 1
Enter Push Element: 5
1=Push 2=Pop 3=Display 4=Exit
Select Option: 3
5
3
1=Push 2=Pop 3=Display 4=Exit
Select Option: 2
Pop element = 5
1=Push 2=Pop 3=Display 4=Exit
Select Option: 3
3
1=Push 2=Pop 3=Display 4=Exit
Select Option: 4
Program Exited%

Lab program 2:

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)

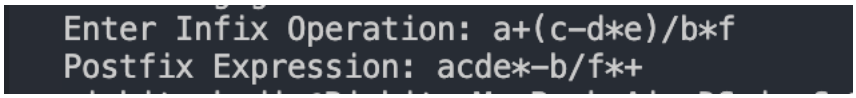
```
#include <stdio.h>
#define MAX 100
int stack[MAX];
int top = -1;

void push(char ele)
{
   if (top==(MAX-1))
      printf("Stack Overflow");
   else
   {
      top = top + 1;
      stack[top] = ele;
   }
}

char pop() //fu'n need to be char
{
   if (top==-1)
      printf("Stack Underflow ");
   else
   {
      int popele = stack[top];
```

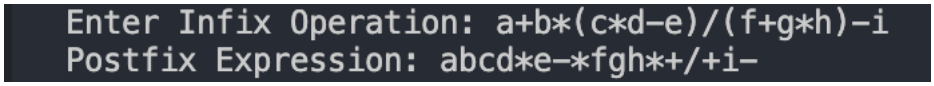
```
        // printf("Pop element = %d", popele);
        top--;
        return popele;
    }
}

int prece(char ch)
{
    if (ch=='*' || ch=='/')
        return 2;
    else if (ch=='+' || ch=='-')
        return 1;
    else
        return 0;
}

int operand(char op)
{
    if ((op>='a' && op<='z') || (op>='A' && op<='Z'))
        return op;
    return '\0';
}

void inToPos (char ifix[], char pfix[])
{
    int i = 0, j = 0;
    char ch;
    while (ifix[i]!='\0')
```

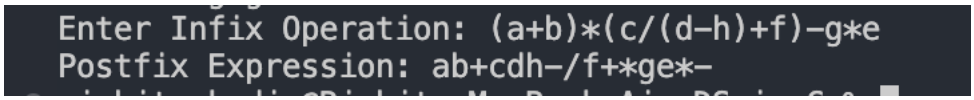
```
{
    ch = ifix[i];
    if (operand(ch))
        pfix[j++] = ch;
    else if (ch =='(')
        push(ch);
    else if (ch ==')')
    {
        while (stack[top]!='(')
        {
            pfix[j++] = pop();
        }
        pop(); //To remove (
    }
    else
    {
        // Very IMP
        while (top != -1 && prece(stack[top])>=prece(ch))
        {
            pfix[j++] = pop();
        }
        push(ch);
    }
    i++;
}
while(top!=-1)
{
    pfix[j++] = pop();
```

```
    }
    pfix[j] = '\0'; //to end the postfix string
}

int main()
{
    char infix[MAX], postfix[MAX];
    printf("Enter Infix Operation: ");
    // gets(infix);
    fgets(infix, MAX, stdin);
    // scanf("%s", infix);
    inToPos(infix, postfix);
    printf("Postfix Expression: %s", postfix);
}
```

Output:







Lab program 3(a):

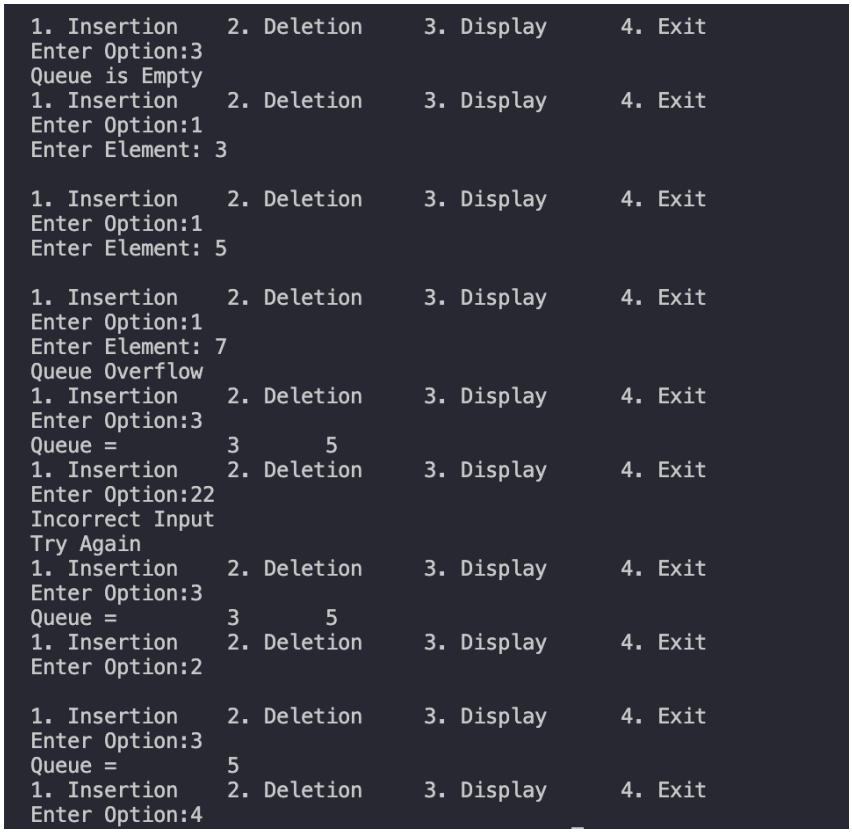
WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>
#define MAX 2

int queue[MAX];
int front=-1, rear=-1;

void insertion(int ele)
{
   if (rear == MAX-1)
      printf("Queue Overflow");
   else
   {
      queue[++rear]=ele;
   }
}
void deletion()
{
   int del;
   if (rear == -1)
      printf("Queue is Empty");
   else
      del=queue[++front];
}
void display()
{
   int i = front;
   if (rear == -1)
      printf("Queue is Empty");
   else
```

```
    {
        printf("Queue = ");
        while (i != rear)
        {
            printf("\t%d", queue[++i]);
        }
    }
}

int main()
{
    int opt, no;
    while (1)
    {
        printf("\n1. Insertion\t2. Deletion\t3. Display\t4. Exit\nEnter Option:");
        scanf("%d", &opt);
        switch (opt)
        {
            case 1:
                printf("Enter Element: ");
                scanf("%d", &no);
                insertion(no);
                break;
            case 2:
                deletion();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Incorrect Input\nTry Again");
```

```
    }
  }
}
```

Output:


1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:3
Queue is Empty
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:1
Enter Element: 3
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:1
Enter Element: 5
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:1
Enter Element: 7
Queue Overflow
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:3
Queue = 3 5
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:22
Incorrect Input
Try Again
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:3
Queue = 3 5
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:2
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:3
Queue = 5
1. Insertion 2. Deletion 3. Display 4. Exit
Enter Option:4

Lab program 3(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>


void insert(int *queue, int *front ,int *rear, int value, int MAX)
{
   if ((*front == 0 && *rear == (MAX -1)) || (*rear == *front -1))
   // if ((*rear + 1) % MAX == *front)
   {
      printf("Queue Overflow");
      return ;
   }
   else if (*front == -1){
      *front = *rear = 0;
      // queue[*rear]=value;
   }
   else if ((*rear == MAX -1 && *front != 0)){
      *rear =0;
      // queue[*rear]=value;
   }
   else{
      (*rear)++;
      // queue[*rear]=value;
   }
   queue[*rear]=value;
}
void delete (int *queue, int *front ,int *rear, int MAX){
   if (*front ==-1){
      printf("Queue Underflow");
```

```
    }
    else if (*front == *rear){
        *front = *rear=-1;
    }
    else if (*front == MAX-1){
        *front=0;
    }
    else{
        (*front)++;
    }
}
void display (int *queue, int front ,int rear, int MAX){
    if (front == -1){
        printf("Queue Underflow");
        return;
    }
    printf("Queue = ");
    if (rear >= front){
        for (int i=front; i<=rear;i++){
            printf("%d\t", queue[i]);
        }
    }
    else{
        for (int j=front; j<MAX; j++){
            printf("%d\t", queue[j]);
        }
        for (int k=0; k<=rear; k++){
            printf("%d\t", queue[k]);
        }
    }
}
```

```
int main()
{
   int MAX = 2;
   int queue[MAX];
   int fr=-1, re=-1;
   int opt, no;
   while (1)
   {
      printf("\n1. Insertion\t2. Deletion\t3. Display\t4. Exit\nEnter Option:");
      scanf("%d", &opt);
      switch (opt)
      {
         case 1:
         {
            printf("Enter Element: ");
            scanf("%d", &no);
            insert(queue, &fr, &re, no, MAX);
            break;
         }
         case 2:
         {
            delete(queue, &fr, &re, MAX);
            break;
         }
         case 3:
         {
            display(queue, fr, re, MAX);
            break;
         }
         case 4:
            return 0;
         default:
```

```
            printf("Incorrect Input\nTry Again");
        }
    }
}
```

Output:

```
1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:2
Queue Underflow
1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:1
Enter Element: 3

1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:1
Enter Element: 5

1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:1
Enter Element: 7
Queue Overflow
1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:3
Queue = 3       5
1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:2

1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:1
Enter Element: 9

1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:3
Queue = 5       9
1. Insertion    2. Deletion     3. Display      4. Exit
Enter Option:4
```

Lab Program-4 (a):

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.

c) Display the contents of the linked list.

AND

Lab Program-5 (a):

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.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};

void insertBeg(struct Node **head, int val){
    struct Node *new = (struct Node *) malloc(sizeof(struct Node));
    new->data=val;
    new->next=*head;
    *head = new;
}

void insertEnd(struct Node **head, int val){
    struct Node *new = (struct Node *) malloc(sizeof(struct Node));
    new->data=val;
    new->next=NULL;
    if (*head == NULL){
        *head=new;
```

```
        return;
    }
    struct Node *temp = *head;
    while (temp->next!=NULL){
        temp = temp->next;
    }
    temp->next=new;
}

void insertPos(struct Node **head, int val, int pos){
    struct Node *new = (struct Node *) malloc(sizeof(struct Node));
    new->data=val;
    if (pos==0){
        new->next=*head;
        *head = new;
        return;
    }
    struct Node *temp = *head;
    for (int i=0; temp->next!=NULL && i < pos-1; i++){
        temp = temp->next;
    }
    if (temp == NULL){
        printf("Out of Range");
        return;
    }
    new->next=temp->next;
    temp->next = new;
}

void deleteBeg(struct Node **head){
    if (*head == NULL){
        printf("Empty LL");
```

```
        return;
    }
    struct Node *temp = *head;
    *head = temp->next;
    free(temp);
}

void deleteEnd(struct Node **head){
    if (*head == NULL){
        printf("Empty LL");
        return;
    }
    struct Node *temp = *head;
    if(temp->next == NULL){
        free(temp);
        *head = NULL;
        return;
    }
    while (temp->next->next!=NULL){
        temp = temp->next;
    }
    free(temp->next);
    temp->next=NULL;
}

void deleteVal(struct Node **head, int val){
    if (*head == NULL){
        printf("Empty LL");
        return;
    }
    struct Node *temp = *head;
    if (temp->data == val){
```

```
        *head = temp->next;
        free(temp);
        return;
    }
    while (temp->next->data!=val){
        temp = temp->next;
    }
    if(temp->next == NULL){
        printf("Value not present in LL");
        free(temp);
        // return;
    }
    struct Node *toDelete = temp->next;
    temp->next = toDelete->next; // Bypass the node to be deleted
    free(toDelete);
}

void display(struct Node **head){
    struct Node *temp = *head;
    if (*head == NULL){
        printf("Empty LL");
        return;
    }
    printf("Queue = ");
    while (temp!=NULL){
        printf("\t%d", temp->data);
        temp = temp->next;
    }

}

int main(){
```

```
    struct Node *head = NULL;
    int choice, data, pos;
    printf("\n1. Insert at beginning\t2. Insert at end\t3. Insert at position\n4. Delete from beginning\t5. Delete from end\t6. Delete by Value\n7. Display\t8. Exit");
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &data);
                insertBeg(&head, data);
                break;
            case 2:
                printf("Enter value: ");
                scanf("%d", &data);
                insertEnd(&head, data);
                break;
            case 3:
                printf("Enter value & position: ");
                scanf("%d %d", &data, &pos);
                insertPos(&head, data, pos);
                break;
            case 4:
                deleteBeg(&head);
                break;
            case 5:
                deleteEnd(&head);
                break;
            case 6:
                printf("Enter value to delete: ");
```

```
            scanf("%d", &pos);
            deleteVal(&head, pos);
            break;
        case 7:
            display(&head);
            break;
        case 8:
            return 0;
        default:
            printf("Invalid choice!\n");
      }
   }
   return 0;
}
```

Output:

```
1. Insert at beginning  2. Insert at end        3. Insert at position
4. Delete from beginning        5. Delete from end      6. Delete by Value
7. Display      8. Exit
Enter your choice: 4
Empty LL
Enter your choice: 1
Enter value: 3

Enter your choice: 2
Enter value: 5

Enter your choice: 3
Enter value & position: 7 1

Enter your choice: 7
Queue =         3       7       5
Enter your choice: 5

Enter your choice: 6
Enter value to delete: 3

Enter your choice: 7
Queue =         7
Enter your choice: 8
```

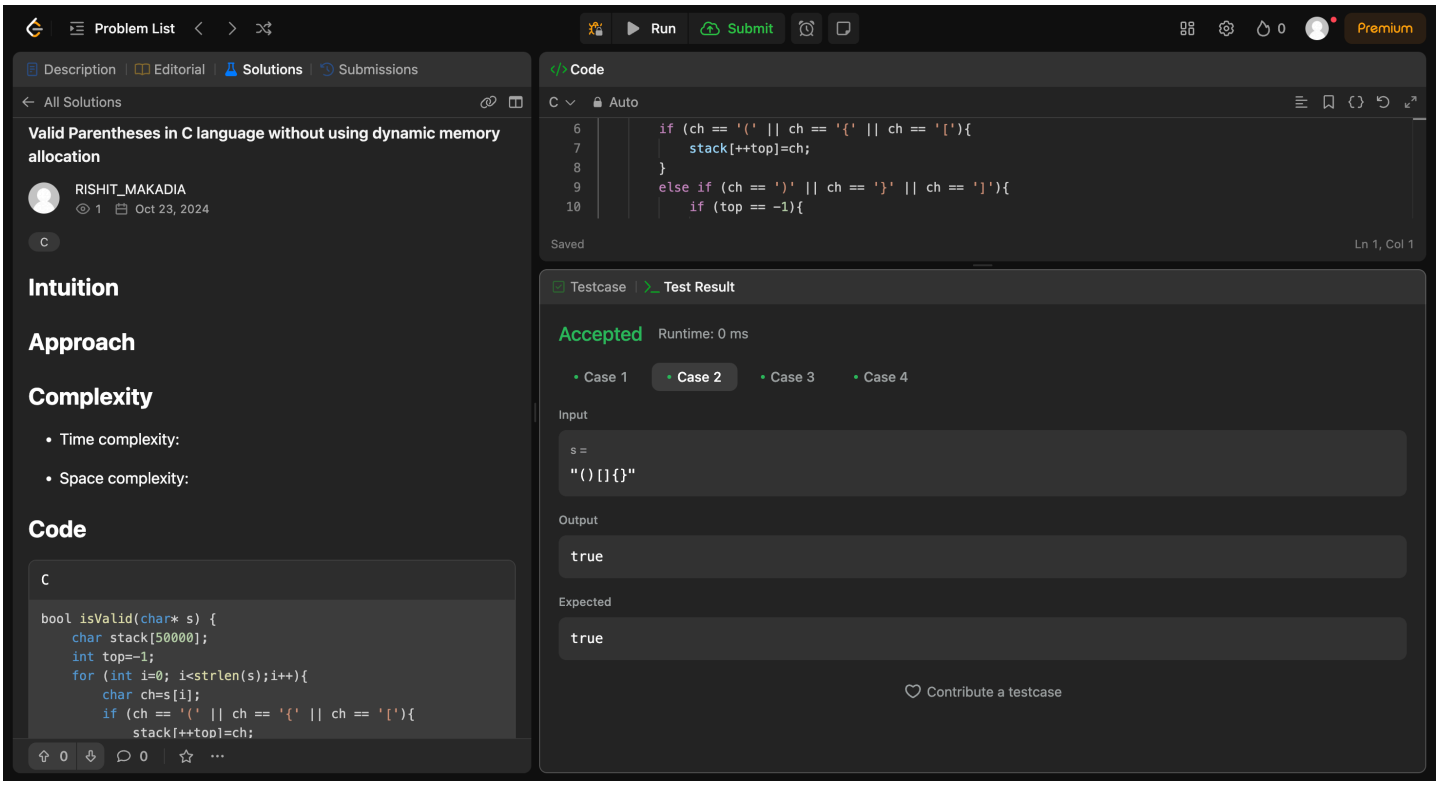
Lab Program-4(b):

Program - Leetcode platform
Valid Parantheses

```
bool isValid(char* s) {
    char stack[50000];
    int top=-1;
    for (int i=0; i<strlen(s);i++){
        char ch=s[i];
        if (ch == '(' || ch == '{' || ch == '['){
            stack[++top]=ch;
        }
        else if (ch == ')' || ch == '}' || ch == ']'){
            if (top == -1){
                return 0;
            }
            char val = stack[top];
            if ((ch == ')' && val == '(') || (ch == '}' && val == '{') || (ch == ']' && val == '[')){
                top--;
            }
            else{
                return 0;
            }
        }
        else{
            return 0;
        }
    }
    if (top != -1){
        return 0;
    }
```

```
    else{
        top = -1;
        return 1;
    }
}
```

Output:



Lab Program-5(b):

Program - Leetcode platform

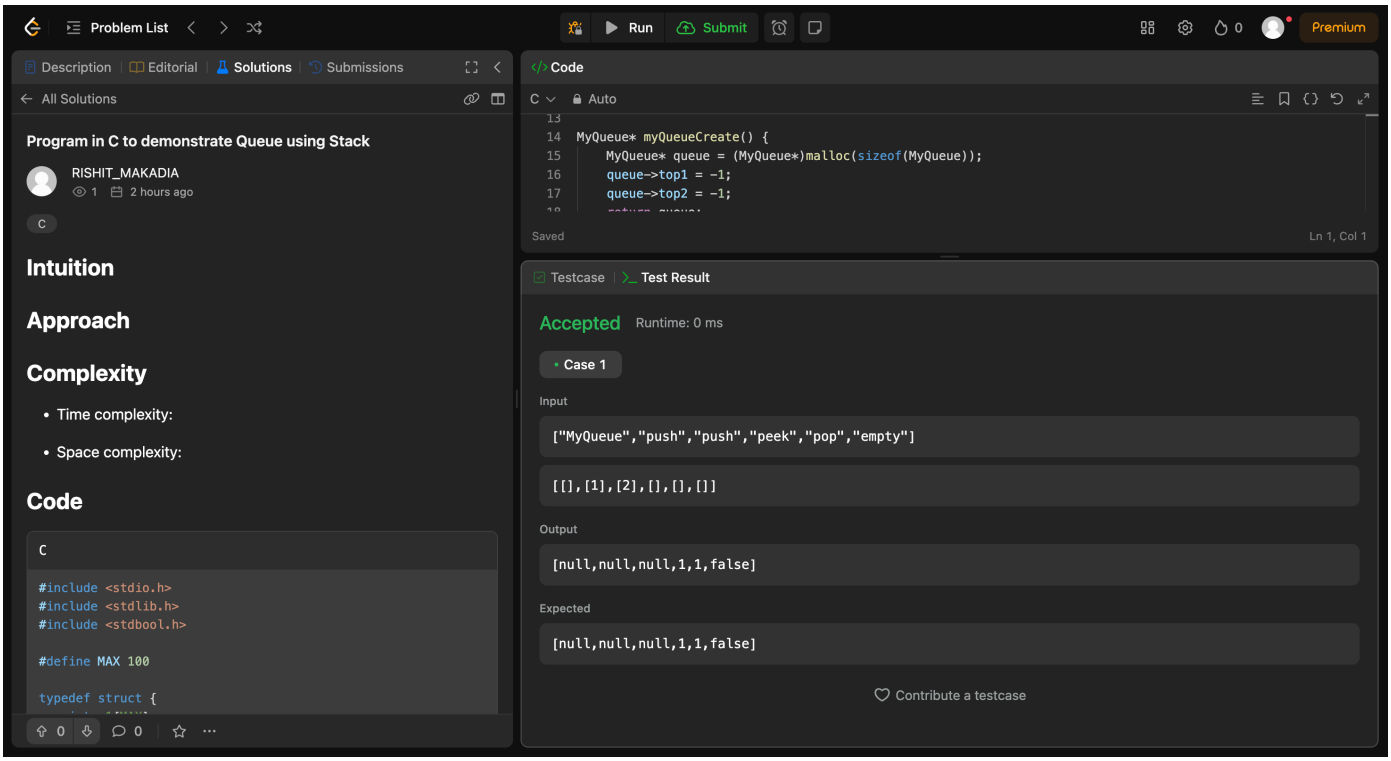
Queue using Stack

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

#define MAX 100

typedef struct {
    int s1[MAX];
    int s2[MAX];
    int top1;
    int top2;
} MyQueue;

MyQueue* myQueueCreate() {
    MyQueue* queue = (MyQueue*)malloc(sizeof(MyQueue));
    queue->top1 = -1;
    queue->top2 = -1;
    return queue;
}

void myQueuePush(MyQueue* obj, int x) {
    if (obj->top1 == MAX - 1) {
        printf("Queue is full\n");
        return;
    }
    obj->s1[++(obj->top1)] = x;
}
```

```
int myQueuePop(MyQueue* obj) {
    if (obj->top2 == -1) {
        while (obj->top1 != -1) {
            obj->s2[++(obj->top2)] = obj->s1[(obj->top1)--];
        }
    }
    if (obj->top2 == -1) {
        printf("Queue is empty\n");
        return -1;
    }
    return obj->s2[(obj->top2)--];
}

int myQueuePeek(MyQueue* obj) {
    if (obj->top2 == -1) {
        while (obj->top1 != -1) {
            obj->s2[++(obj->top2)] = obj->s1[(obj->top1)--];
        }
    }
    if (obj->top2 == -1) {
        printf("Queue is empty\n");
        return -1;
    }
    return obj->s2[obj->top2];
}

bool myQueueEmpty(MyQueue* obj) {
    return obj->top1 == -1 && obj->top2 == -1;
}

void myQueueFree(MyQueue* obj) {
    free(obj);
```

```
}

/**
 * Your MyQueue struct will be instantiated and called as such:
 * MyQueue* obj = myQueueCreate();
 * myQueuePush(obj, x);
 * int param_2 = myQueuePop(obj);
 * int param_3 = myQueuePeek(obj);
 * bool param_4 = myQueueEmpty(obj);
 * myQueueFree(obj);
 */
```

Output:



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>

typedef struct Node {
   int data;
   struct Node *next;
}Node;

Node *create(int val)
{
    Node *newN = (struct Node *) malloc(sizeof(Node));
    newN->data=val;
    newN->next=NULL;
    return newN;
}
Node *insertBeg(Node *head,int val)
{
   Node *new=create(val);
   new->next=head;
   head=new;
   return head;
}

Node *concate(Node *head1, Node *head2)
{
   Node *temp=head1;
   while (temp->next != NULL)
      temp=temp->next;
```

```
    temp->next=head2;
    return head1;
}

Node *sort(Node *head)
{
    int t;
    Node *temp, *curr;
    curr=head;
    while (curr!=NULL)
    {
        temp=head;
        while(temp->next!=NULL)
        {
            if ((temp->data) > (temp->next->data))
            {
                t=temp->data;
                temp->data = temp->next->data;
                temp->next->data = t;
            }
            temp=temp->next;
        }
        curr=curr->next;
    }
    return head;
}

Node *reverse(Node *head)
{
    Node *temp, *new, *prev=NULL;
    temp = head;
    while (temp!=NULL)
```

```
    {
        new=temp->next;
        temp->next=prev;
        prev=temp;
        temp=new;
    }
    return prev;
}

void display(Node *head)
{
    struct Node *temp=head;
    while(temp!=NULL){
        printf("%d ", temp->data);
        temp=temp->next;
    }
    printf("\n");
}

int main()
{
    Node *l1 = NULL;
    Node *l2 = NULL;

    l1 = insertBeg(l1, 5);
    l1 = insertBeg(l1, 3);
    l1 = insertBeg(l1, 7);

    l2 = insertBeg(l2, 4);
    l2 = insertBeg(l2, 8);
    l2 = insertBeg(l2, 6);
```

```
    printf("List 1 = ");
    display(l1);
    printf("List 2 = ");
    display(l2);

    l1 = sort(l1);
    l2 = sort(l2);

    l2 = reverse(l2);

    printf("Sorted List 1 = ");
    display(l1);
    printf("Sorted and then Reversed List 2 = ");
    display(l2);

    printf("List 1 + List 2 = ");
    l1=concate(l1, l2);
    display(l1);
}
```

Output:

```
List 1 = 7 3 5
List 2 = 6 8 4
Sorted List 1 = 3 5 7
Sorted and then Reversed List 2 = 8 6 4
List 1 + List 2 = 3 5 7 8 6 4
```

Lab Program-6(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 *top=NULL, *front=NULL, *rear=NULL;

void push(int val)
{
    struct Node *new = (struct Node *) malloc(sizeof(struct Node));
    new->data=val;
    new->next=top;
    top=new;
}
void pop()
{
    struct Node *temp;
    if (top==NULL)
        printf("Stack Underflow ");
    else
    {
        temp=top;
        top=top->next;
        free(temp);
    }
}
void displayStack()
```

```
{
   struct Node *temp=top;
   while(temp!=NULL){
      printf("%d ", temp->data);
      temp=temp->next;
   }
}

void enQueue(int val)
{
   struct Node *new = (struct Node *) malloc(sizeof(struct Node));
   new->data=val;
   new->next=NULL;
   if (rear==NULL && front==NULL)
      rear=front=new;
   else
   {
      rear->next=new;
      rear=new;
   }
}
void deQueue()
{
   struct Node *temp;
   if (front==NULL)
      printf("Queue Underflow\n ");
   else
   {
      temp=front;
      front=front->next;
      if (front==NULL)
         rear=NULL;
```

```
        free(temp);
    }
}
void displayQueue()
{
    struct Node *temp=front;
    while(temp!=NULL){
        printf("%d ", temp->data);
        temp=temp->next;
    }
}


int main()
{
 int choice, data;
    printf("\n1. Stack PUSH\t2. Stack POP\t3. Display Stack\n4. Queue Insertion\t5. Queue Deletion\t6. Display Queue\n7. Exit");
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &data);
                push(data);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
```

```
            break;
        case 4:
            printf("Enter value: ");
            scanf("%d", &data);
            enQueue(data);
            break;
        case 5:
            deQueue();
            break;
        case 6:
            displayQueue();
            break;
        case 7:
            return 0;
        default:
            printf("Invalid choice!\n");
      }
   }
   return 0;
}
```

Output:

```
1. Stack PUSH   2. Stack POP    3. Display Stack
4. Queue Insertion      5. Queue Deletion       6. Display Queue
7. Exit
Enter your choice: 2
Stack Underflow
Enter your choice: 1
Enter value: 3

Enter your choice: 1
Enter value: 5

Enter your choice: 1
Enter value: 7

Enter your choice: 2

Enter your choice: 3
5 3
Enter your choice: 5
Queue Underflow

Enter your choice: 4
Enter value: 2

Enter your choice: 4
Enter value: 4

Enter your choice: 4
Enter value: 6

Enter your choice: 5

Enter your choice: 6
4 6
Enter your choice: 7
```

Lab program-7 (a):

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>

struct Node
{
   int data;
   struct Node *next;
   struct Node *prev;
};
struct Node *create(int val)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   new->data = val;
   new->next = NULL;
   new->prev = NULL;
   return new;
}

void createList(struct Node **head, int val){
   *head=create(val);
}

void insertLeft (struct Node **head, int find, int val){
   if ((*head)==NULL){
```

```
        printf("No LL Exist");
        return;
    }
    struct Node *temp=*head;
    struct Node *new=create(val);
    while(temp!=NULL && temp->data!=find){
        temp=temp->next;
    }
    if (temp==NULL){
        printf("No Node found with value %d", find);
        return;
    }
    new->next=temp;
    new->prev=temp->prev;
    if (temp->prev!=NULL)
        temp->prev->next=new;
    else
        (*head)=new;
    temp->prev=new;
}

void deleteVal (struct Node **head, int del){
    if ((*head)==NULL){
        printf("LL doesn't exist");
        return;
    }
    struct Node *temp=*head;
    while(temp!=NULL && temp->data!=del){
        temp=temp->next;
    }
    if (temp==NULL){
        printf("No Node found with value %d", del);
```

```
        return;
    }
    if (temp->prev!=NULL)
        temp->prev->next=temp->next;
    else
        (*head)=temp->next;
    if (temp->next!=NULL)
        temp->next->prev=temp->prev;
    free(temp);
}

void display(struct Node *head){
    struct Node *temp=head;
    while(temp!=NULL){
        printf("%d\t", temp->data);
        temp=temp->next;
    }
}

int main(){
    struct Node *head = NULL;
    int choice, data, pos;
    printf("\n1. Create a new LL\t2. Insert Left to\t3. Delete by value\t4. Display\t5. Exit\t\n");
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter value: ");
                scanf("%d", &data);
```

```
            createList(&head, data);
            break;
        case 2:
            printf("Enter value: ");
            scanf("%d", &data);
            printf("Enter to the left of value: ");
            scanf("%d", &pos);
            insertLeft(&head, pos, data);
            break;
        case 3:
            printf("Enter value to delete: ");
            scanf("%d", &data);
            deleteVal(&head, data);
            break;
        case 4:
            display(head);
            break;
        case 5:
            return 0;
        default:
            printf("Invalid choice!\n");
      }
   }
}
```

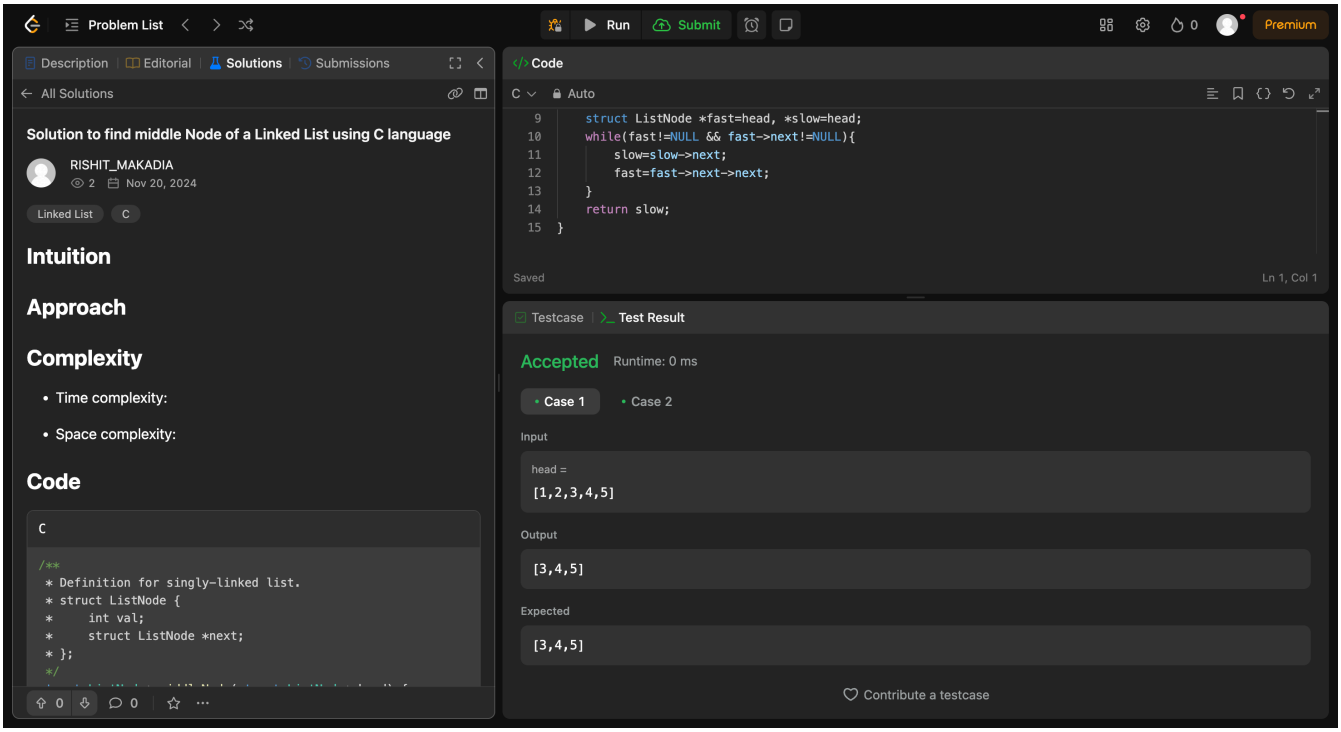
Output:

```
1. Create a new LL      2. Insert Left to       3. Delete by value      4. Display      5. Exit

Enter your choice: 3
Enter value to delete: 7
LL doesn't exist
Enter your choice: 1
Enter value: 3

Enter your choice: 2
Enter value: 5
Enter to the left of value: 3

Enter your choice: 4
5       3
Enter your choice: 3
Enter value to delete: 3

Enter your choice: 4
5
Enter your choice: 5
```

Lab Program-7(b):
Program - Leetcode platform
Middle element of the Linked List

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* middleNode(struct ListNode* head) {
    struct ListNode *fast=head, *slow=head;
    while(fast!=NULL && fast->next!=NULL){
        slow=slow->next;
        fast=fast->next->next;
    }
    return slow;
}
```

Output:



Lab program-8 (a):

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;
   struct Node *right;
}Node;

Node *create(Node* root, int val){
   Node *new = (Node *)malloc(sizeof(Node));
   new->data = val;
   new->left = NULL;
   new->right = NULL;
   return new;
}

Node *insert(Node* root, int val){
   if (root == NULL)
      return create(root, val);
   else if (val<(root->data))
      root->left = insert(root->left, val);
   else if (val>(root->data))
```

```
        root->right = insert(root->right, val);
    return root;
}

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

int main()
{
    Node *root = NULL;
    int data, no;
    printf("Enter a new Binary Search Tree\n");
    printf("Enter No. of Nodes: ");
```

```
    scanf("%d", &no);
    printf("Enter value: ");
    for(int i=0; i<no;i++){
        scanf("%d", &data);
        if (i==0)
            root=create(root, data);
        else
            insert(root, data);
    }
    printf("\nPre-Order : ");
    preOrder(root);
    printf("\nIn-Order : ");
    inOrder(root);
    printf("\nPost-Order : ");
    postOrder(root);
}
```

Output:

```
Enter a new Binary Search Tree
Enter No. of Nodes: 9
Enter value: 5 4 1 7 11 8 2 3 10

Pre-Order :  5 4 1 2 3 7 11 8 10
In-Order :  1 2 3 4 5 7 8 10 11
Post-Order :  3 2 1 4 10 8 11 7 5
```

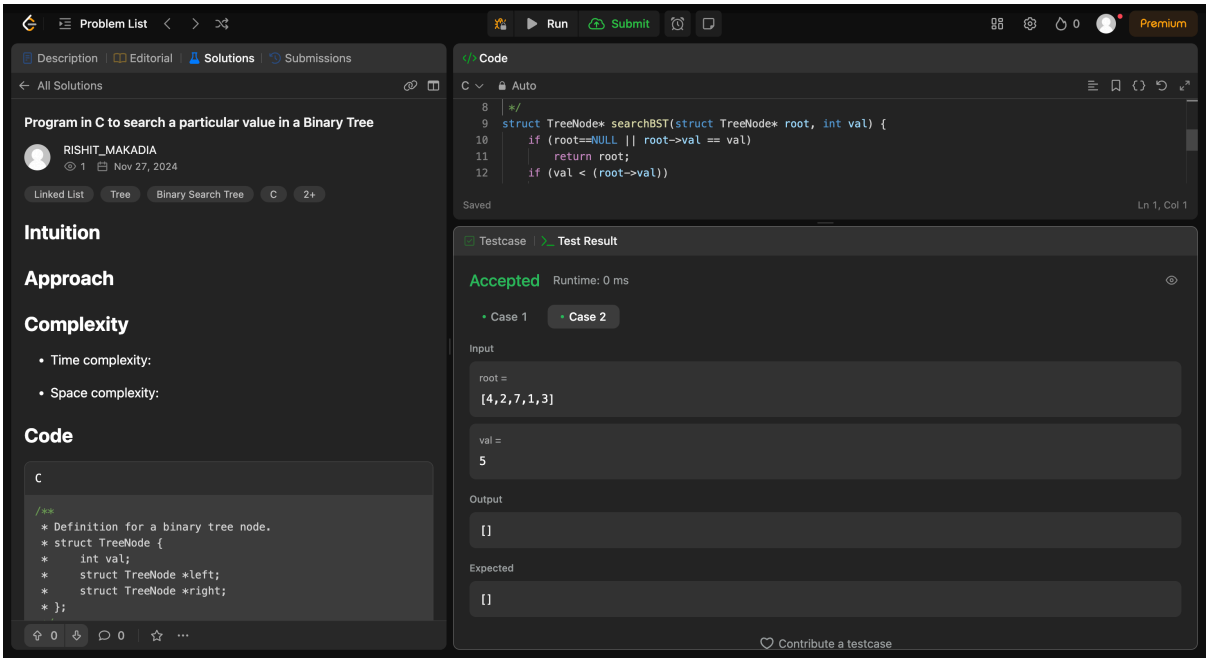
Lab Program-8(b):

Program - Leetcode platform

Search in Binary Search Tree

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* searchBST(struct TreeNode* root, int val) {
    if (root==NULL || root->val == val)
        return root;
    if (val < (root->val))
        return searchBST(root->left, val);
    else
        return searchBST(root->right, val);
}
```

Output:


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
All Solutions
Program in C to search a particular value in a Binary Tree
RISHIT_MAKADIA
1
Nov 27, 2024
Linked List
Tree
Binary Search Tree
C
2+
Intuition
Approach
Complexity
Time complexity:
Space complexity:
Code
C
/**
* Definition for a binary tree node.
* struct TreeNode {
*   int val;
*   struct TreeNode *left;
*   struct TreeNode *right;
* };
Code
C
Auto
*/
struct TreeNode* searchBST(struct TreeNode* root, int val) {
if (root==NULL || root->val == val)
return root;
if (val < (root->val))
Saved
Ln 1, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[4,2,7,1,3]
val =
5
Output
[]
Expected
[]
Contribute a testcase

Lab Program-9 (a):

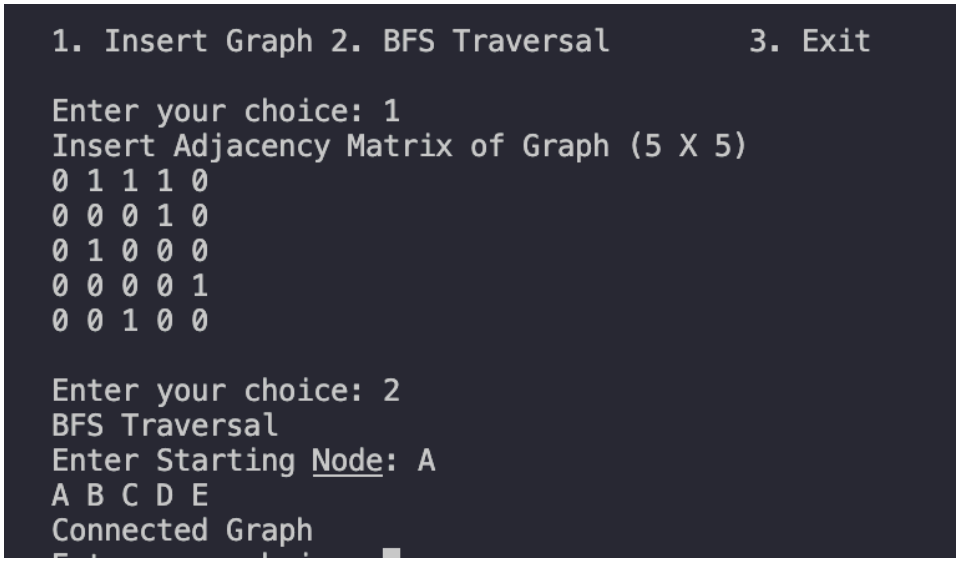
Write a program to traverse a graph using BFS method.

```
#include <stdio.h>
#define MAX 5

void bfs(int adjM[MAX][MAX], int visited[MAX], int start)
{
   int queue[MAX], rear = -1, front = -1, count=1;
   for (int i = 0; i < MAX; i++)
   {
      visited[i] = 0;
   }
   queue[++rear] = start;
   front++;
   visited[start] = 1;
   while (rear >= front)
   {
      start = queue[front++];
      printf("%c ", (start + 65));
      for (int k = 0; k < MAX; k++)
      {
         if (adjM[start][k] == 1 && visited[k] == 0)
         {
            queue[++rear] = k;
            visited[k] = 1;
            count++;
         }
      }
   }
   if(count==MAX)
```

```
        printf("\nConnected Graph");
    else
        printf("\nNot-Connected Graph");
}

int main(){
    int visited[MAX]={0}, adj[MAX][MAX];
    int choice;
    char ch;
    printf("\n1. Insert Graph\t2. BFS Traversal\t3. Exit\t\n");
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice){
            case 1:
                printf("Insert Adjacency Matrix of Graph (5 X 5)\n");
                for(int i=0; i<MAX; i++){
                    for(int j=0; j<MAX; j++){
                        scanf("%d", &adj[i][j]);
                    }
                }
                break;
            case 2:
                printf("BFS Traversal\n");
                printf("Enter Starting Node: ");
                scanf(" %c", &ch);
                bfs(adj, visited, (ch-65));
                break;
            case 3:
                return 0;
                break;
            default:
```

```
            printf("Invalid Option");
        }
    }
}
```

Output:



Lab Program-9 (b):

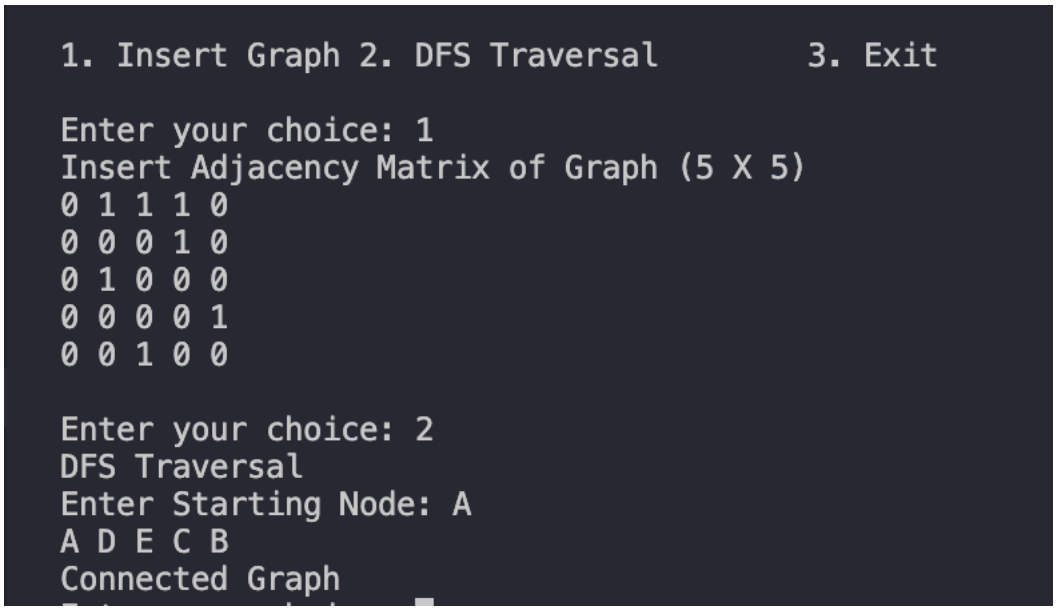
Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>
#define MAX 5

void dfs(int adjM[MAX][MAX], int visited[MAX], int start)
{
   int stack[MAX], top = -1, count=1;
   for (int i = 0; i < MAX; i++)
   {
      visited[i] = 0;
   }
   stack[++top] = start;
   visited[start] = 1;
   while (top != -1)
   {
      start = stack[top--];
      printf("%c ", (start + 65));
      for (int k = 0; k < MAX; k++)
      {
         if (adjM[start][k] == 1 && visited[k] == 0)
         {
            stack[++top] = k;
            visited[k] = 1;
            count++;
         }
      }
   }
   if(count==MAX)
      printf("\nConnected Graph");
```

```
    else
        printf("\nNot-Connected Graph");
}
int main(){
    int visited[MAX]={0}, adj[MAX][MAX];
    int choice;
    char ch;
    printf("\n1. Insert Graph\t2. DFS Traversal\t3. Exit\t\n");
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice){
            case 1:
                printf("Insert Adjacency Matrix of Graph (5 X 5)\n");
                for(int i=0; i<MAX; i++){
                    for(int j=0; j<MAX; j++){
                        scanf("%d", &adj[i][j]);
                    }
                }
                break;
            case 2:
                printf("DFS Traversal\n");
                printf("Enter Starting Node: ");
                scanf(" %c", &ch);
                dfs(adj, visited, (ch-65));
                break;
            case 3:
                return 0;
                break;
            default:
                printf("Invalid Option");
        }
```

```
    }
}
```

Output:


1. Insert Graph 2. DFS Traversal 3. Exit
Enter your choice: 1
Insert Adjacency Matrix of Graph (5 X 5)
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
Enter your choice: 2
DFS Traversal
Enter Starting Node: A
A D E C B
Connected Graph

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-> L as H(K)=K mod 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.

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

int key[20], n, m;
int *ht;
int count = 0;

void insert(int key) {
   int index = key % m;
   while (ht[index] != -1) {
      index = (index + 1) % m;
   }
   ht[index] = key;
   count++;
}

void display() {
   int i;
   if (count == 0) {
      printf("\nHash Table is empty\n");
      return;
   }
```

```
    printf("\nHash Table:\n");
    for (i = 0; i < m; i++) {
        printf("T[%d] = %d\n", i, ht[i]);
    }
}

int main() {
    int i;
    printf("Enter No. of Employee: ");
    scanf("%d", &n);

    printf("Enter Size of Hash Table: ");
    scanf("%d", &m);

    ht = (int *)calloc(m, sizeof(int));
    for (i = 0; i < m; i++)
        ht[i] = -1;

    printf("Enter 4-digit key values for %d Employee Records:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &key[i]);
    }

    for (i = 0; i < n; i++) {
        if (count == m) {
            printf("\nHash table is full");
            break;
        }
        insert(key[i]);
    }

    display();
```

```
    free(ht);
    return 0;
}
```

Output: