VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Vyom Gupta (1BM22CS333)

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
Dec 2023- March 2024
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 **Vyom Gupta (1BM22CS333)**, 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 2023-24. 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. Lakshmi Neelima
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
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>
#include<stdlib.h>
#define SIZE 10
int stack[SIZE],top=-1;
```

```
void push(int value)
{
   if(top==SIZE-1)
      printf("Overflow\n");
   else{
      top=top+1;
   stack[top]=value;
   printf("%d inserted\n",value);
         }
}

void pop()
{
   int value;
   if(top==-1)
       printf("Underflow\n");
   else{
      value=stack[top];
      printf("Popped element: %d\n",value);
      top=top-1;
         }
}
void display()
{
   int i;
   if(top==-1)
     printf("Stack is empty");
   else{
         printf("The elements in the stack are:\n");
         for(i=0;i<=top;i++)
         printf("%d     ",stack[i]);
         }
}

void main()
{
   int value,choice;
   while(1){
      printf("\n1:Push\n2.Pop\n3.Display\n4.Exit\n");
      printf("Enter your choice:");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1: printf("\nEnter the value: ");
```

```
                scanf("%d",&value);
                push(value);
                break;
        case 2: pop();
                break;
        case 3: display();
                break;
        case 4: exit(0);
                break;
        default: printf("Invalid input\n");
      }
   }
}
```

OUTPUT:

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

Enter the value: 23
23 inserted

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

Enter the value: 45
45 inserted

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

Enter the value: 78
78 inserted
```

```
1:Push
2.Pop
3.Display
4.Exit
Enter your choice: 2
Popped element: 78

1:Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
The stack elements are:
23   45
1:Push
2.Pop
3.Display
4.Exit
Enter your choice: 4
```

LAB PROGRAM 2:

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

```
#include<stdio.h>

#include<string.h>

int index1=0,top=-1,pos=0,length;

char symbol,temp,infix[20],postfix[20],stack[20];

void infixtopostfix();

void push(char symbol);

char pop();

int precedence(char symbol);

void main()

{

   printf("Enter the infix expression:");

   scanf("%s",infix);
```

```
    infixtopostfix();
    printf("\nInfix expression:%s",infix);
    printf("\nPostfix expression:%s",postfix);
}
void infixtopostfix(){
length=strlen(infix);
 push('#');
 while(index1<length){
     symbol=infix[index1];
     switch(symbol)
   {
         case '(':   push (symbol);
                     break;
         case ')':   temp=pop();
                     while (temp !='(')
                     {
                             postfix[pos]=temp;
                             pos++;
                             temp=pop();
                     }
                     break;
         case '+':
         case '-':
         case '*':
         case '/':
            while(precedence(stack[top])>=precedence(symbol))
            {
               temp=pop();
               postfix[pos++]=temp;
```

```
            }
            push(symbol);
            break;
        default:postfix[pos++]=symbol;
            }
     Index1++;
}
 while(top >0) {
     temp=pop();
     postfix[pos++]=temp; }
}
void push(char symbol)
{
    top=top+1;
    stack[top]=symbol;
}
char pop()
{
    char symbol;
    symbol=stack[top];
    top=top-1;
    return (symbol);
}
int precedence(char symbol)
{
    int p;
 switch(symbol){
        case '*':
        case'/':     p=2;
```

```
                    break;
        case '+':
        case '-':   p=1;
                    break;
        case '(':   p=0;
                    break;
        case '#':   p=-1;
                    break;
          }
   return(p);
}
```

OUTPUT:

```
Enter the infix expression:a+b+c+d-e

Infix expression:a+b+c+d-e
Postfix expression:ab+c+d+e-
```

LAB PROGRAM 3:

a) Write a program 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>

#include <stdlib.h>

#define MAX 10

int queue[MAX];

int front = -1, rear = -1;

void insert()

{
```

```
	int item;
	if(rear == MAX-1)
		printf("\n OVERFLOW");
	else
	{
		if(front==-1)
			front=0;
		printf("\n Enter the element to be inserted in the queue : ");
		scanf("%d", &item);
		rear=rear+1;
		queue[rear] = item;
		printf("%d inserted successfully",item);
	}
}
void delete()
{
	int val;
	if(front == -1 || front>rear)
	{
		printf("\n QUEUE UNDERFLOW");
	}
	else
	{
		val = queue[front];
		printf("\n The number deleted is : %d", val);
		front=front+1;
		if(front > rear)
			front = rear = -1;
	}
```

```
}
void display()
{
    int i;
    printf("\n");
    if(front == -1 || front > rear)
        printf("\n QUEUE IS EMPTY");
    else
    {
        for(i = front;i <= rear;i++)
            printf("%d      ", queue[i]);
    }
}
void main()
{
    int choice;
    while(1)
    {
        printf("\nMENU");
        printf("\n 1. Insert an element");
        printf("\n 2. Delete an element");
        printf("\n 3. Display the queue");
        printf("\n 4. EXIT");
        printf("\n Enter your option :");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: insert();
                break;
```

```
                        case 2: delete();
                                break;
                        case 3: display();
                                break;
                        case 4: exit(0);
                                break;
                        default: printf("Invalid input");
                }
        }
}
```

OUTPUT:

```
MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :1

Enter the element to be inserted in the queue : 68
68 inserted successfully

MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :1

Enter the element to be inserted in the queue : 83
83 inserted successfully

MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :1

Enter the element to be inserted in the queue : 90
90 inserted successfully
```

```
MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :2

The number deleted is : 68

MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :3

83      90
MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :4
```

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 10

int queue[SIZE];

int front = -1, rear = -1;

void insert()

{

        int item;

        if((front==rear+1)||(front==0 && rear==SIZE-1))

                printf("\n QUEUE OVERFLOW");

        else
```

```
	{
		if(front==-1)
			front=0;
		printf("\n Enter the element to be inserted : ");
		scanf("%d", &item);
		rear=(rear+1)%SIZE;
		queue[rear] = item;
		printf("%d inserted successfully",item);
	}
}
void delete()
{
	int val;
	if(front == -1)
	{
		printf("\n QUEUE UNDERFLOW");
	}
	else
	{
		val = queue[front];
		if(front==rear)
		{
			front=-1;
			rear=-1;
		}
		else
		{
			front=(front+1)%SIZE;
		}
```

```
            printf("\n The number deleted is : %d", val);
        }
}
void display()
{
        int i;
        printf(" \n");
        if(front == -1)
            printf("\n QUEUE IS EMPTY");
        else
        {
            for(i = front;i !=rear;i=(i+1)%SIZE)
                printf("%d      ", queue[i]);
        }
        printf("%d",queue[i]);
}
void main()
{
        int choice;
        while(1)
        {
            printf("\nMENU");
            printf("\n 1. Insert an element");
            printf("\n 2. Delete an element");
            printf("\n 3. Display the queue");
            printf("\n 4. EXIT");
            printf("\n Enter your option :");
            scanf("%d", &choice);
            switch(choice)
```

```
                {
                        case 1: insert();
                                break;
                        case 2: delete();
                                break;
                        case 3: display();
                                break;
                        case 4: exit(0);
                                break;
                        default: printf("Invalid input");
                }
        }
}
```

OUTPUT:

```
MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :1

Enter the element to be inserted : 37
37 inserted successfully

MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :1

Enter the element to be inserted : 43
43 inserted successfully

MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :1

Enter the element to be inserted : 74
74 inserted successfully
```

```
MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :2

The number deleted is : 37
MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :3

43      74
MENU
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :4
```

LAB PROGRAM 4:

1)

Write a program 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.

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

   int data;

   struct node * next;

};
```

```
struct node *head=NULL,*newnode,*temp;

void create()
{
   int i,n;
   printf(" Enter the number of elements:");
   scanf("%d",&n);

   for(i=0;i<n;i++)
   {
      newnode=(struct node *)malloc(sizeof(struct node));
      printf(" Enter the element %d: ",i+1);
      scanf("%d",&newnode->data);
      newnode->next=NULL;

      if(head==NULL)
      {
         temp=head=newnode;
      }
      else{
         temp->next=newnode;
         temp=newnode;
      }
   }
}
void display()
{
   temp=head;
   printf(" The elements are:\n");
```

```
    while(temp!=NULL)
    {
        printf(" %d\n",temp->data);
        temp=temp->next;
    }
}

void insert_beg()
{
    newnode=(struct node *)malloc(sizeof(struct node));
    printf(" Enter the new element:\n");
    scanf("%d",&newnode->data);
    newnode->next=head;
    head=newnode;
}

void insert_end()
{
    newnode=(struct node *)malloc(sizeof(struct node));
    printf(" Enter the new element:\n");
    scanf("%d",&newnode->data);
    newnode->next=NULL;
    temp=head;
    while(temp->next!=NULL)
    {
        temp=temp->next;
    }
    temp->next=newnode;
}
```

```
void insert_pos()
{
    int pos,i=1;
    newnode=(struct node *)malloc(sizeof(struct node));
    printf(" Enter the position:");
    scanf("%d",&pos);

    if(pos<0)
    {
        printf("Invalid position\n");
    }
    else
    {
        temp=head;
        while(i<pos-1)
        {
            temp=temp->next;
            i++;
        }
        printf(" Enter the new element:");
        scanf("%d",&newnode->data);
        newnode->next=temp->next;
        temp->next=newnode;
    }
}

void main()
{
```

```
    int choice;
    printf(" MENU\n 1.Create\n 2.Insert at beginnning\n 3.Insert at end\n 4.Insert at position\n
5.Display \n 6.Exit\n");
    while(1)
    {
        printf("\n Enter operation:");
        scanf("%d",&choice);
         switch(choice)
         {
                    case 1:create();
                            break;
                    case 2: insert_beg();
                            break;
                    case 3: insert_end();
                            break;
                    case 4:insert_pos();
                            break;
                    case 5: display();
                            break;
                    case 6: exit(0);
                            break;
                    default: printf(" invalid output\n");
           }
        }
}
```

OUTPUT:

```
MENU
1.Create
2.Insert at beginnning
3.Insert at end
4.Insert at position
5.Display
6.Exit

Enter operation:1
Enter the number of elements: 3
Enter the element 1: 2
Enter the element 2: 4
Enter the element 3: 6

Enter operation:2
Enter the new element: 10

Enter operation:3
Enter the new element: 5

Enter operation:4
Enter the position: 2
Enter the new element: 34

Enter operation:5
The elements are:
10      34      2       4       6       5
Enter operation:6
```

2)

Write a program 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;
};

struct node *head=NULL,*newnode,*temp;

void create()
{
    int i,n;
    printf(" Enter the number of elements:");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        newnode=(struct node *)malloc(sizeof(struct node));
        printf(" Enter the element %d: ",i+1);
        scanf("%d",&newnode->data);
        newnode->next=NULL;

        if(head==NULL)
        {
            temp=head=newnode;
        }
        else{
            temp->next=newnode;
            temp=newnode;
        }
```

```
    }
}
void display()
{
    temp=head;
    printf(" The elements are:\n");
    while(temp!=NULL)
    {
        printf(" %d\t",temp->data);
        temp=temp->next;
    }
}

void delete_beg()
{
    temp=head;
    if(head==NULL)
    {
        printf("List is empty\n");
    }
    else
    {
         head=temp->next;
         free(temp);
    }
printf(" Element successfully deleted\n");
}

void delete_end()
```

```
{
    temp=head;
    struct node *prenode;
    while(temp->next!=NULL)
    {
        prenode=temp;
        temp=temp->next;
    }
    if(temp==head)
    {
        head=NULL;
    }
    else
    {
        prenode->next=NULL;

    }
    free(temp);
     printf(" Element successfully deleted\n");
}

void delete_pos()
{
    struct node *nextnode;
    int pos,i=1;
    printf(" Enter the position:\n");
    scanf("%d",&pos);
    temp=head;
    while(i<pos-1)
```

```
    {
        temp=temp->next;
        i++;
    }
    nextnode=temp->next;
    temp->next=nextnode->next;
    free(nextnode);
    printf(" Element successfully deleted\n");
}
void main()
{
    int choice;
    printf(" MENU\n 1.Create\n 2.Insert at beginnning\n 3.Insert at end\n 4.Insert at position\n
5.Display \n 6.Exit\n");
    while(1)
    {
        printf("\n Enter operation:");
        scanf("%d",&choice);
         switch(choice)
         {
                case 1:create();
                        break;
                case 2: delete_beg();
                        break;
                case 3: delete_end();
                        break;
                case 4:delete_pos();
                        break;
                case 5: display();
                        break;
```

```
                case 6: exit(0);
                        break;
                default: printf(" Invalid output\n");
        }
    }
}
```

OUTPUT:

```
MENU
1.Create
2.Insert at beginnning
3.Insert at end
4.Insert at position
5.Display
6.Exit

Enter operation:1
Enter the number of elements:5
Enter the element 1: 2
Enter the element 2: 4
Enter the element 3: 6
Enter the element 4: 8
Enter the element 5: 10

Enter operation:2
Element successfully deleted

Enter operation:3
Element successfully deleted

Enter operation:4
Enter the position: 1
Element successfully deleted

Enter operation:5
The elements are:
4       8
Enter operation:6
```

LAB PROGRAM 5:

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

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

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



void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

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

void sortList(struct Node** head) {
    struct Node *current, *nextNode;
    int temp;
    current = *head;

    while (current != NULL) {
```

```
        nextNode = current->next;
        while (nextNode != NULL) {
            if (current->data > nextNode->data) {
                temp = current->data;
                current->data = nextNode->data;
                nextNode->data = temp;
            }
            nextNode = nextNode->next;
        }
        current = current->next;
    }
}

void reverseList(struct Node** head) {
    struct Node *prev, *current, *nextNode;
    prev = NULL;
    current = *head;

    while (current != NULL) {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    *head = prev;
}

void concatenateLists(struct Node** list1, struct Node* list2) {
    if (*list1 == NULL) {
        *list1 = list2;
        return;
    }
    struct Node* temp = *list1;
    while (temp->next != NULL) {
```

```
        temp = temp->next;
    }
    temp->next = list2;
}

void main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice;
    int data;

    while(1)
    {
        printf("\n1. Insert into List 1\n");
        printf("2. Insert into List 2\n");
        printf("3. Sort List 1\n");
        printf("4. Reverse List 1\n");
        printf("5. Concatenate Lists\n");
        printf("6. Print Lists\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert into List 1: ");
                scanf("%d", &data);
                insertAtBeginning(&list1, data);
                break;
            case 2:
                printf("Enter data to insert into List 2: ");
                scanf("%d", &data);
                insertAtBeginning(&list2, data);
                break;
```

```
            case 3:
                sortList(&list1);
                printf("List 1 sorted.\n");
                break;
            case 4:
                reverseList(&list1);
                printf("List 1 reversed.\n");
                break;
            case 5:
                concatenateLists(&list1, list2);
                printf("Lists concatenated.\n");
                break;
            case 6:
                printf("List 1: ");
                printList(list1);
                printf("List 2: ");
                printList(list2);
                break;
            case 7:
                exit(0);
                break;
            default:
                printf("Invalid choice\n");
        }
    }
}
```

OUTPUT:

```
1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 1
Enter data to insert into List 1: 5

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 1
Enter data to insert into List 1: 3

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 1
Enter data to insert into List 1: 9
```

```
1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 2
Enter data to insert into List 2: 4

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 3
List 1 sorted.

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 4
List 1 reversed.

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 6
```

```
List 1: 9 5 3
List 2: 4

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 5
Lists concatenated.

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 6
List 1: 9 5 3 4
List 2: 4

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 7
```

LAB PROGRAM 6:

Write a program to Implement Single Link List to simulate Stack & Queue Operations.

```
#include <stdio.h>

#include <stdlib.h>

struct node {

    int info;

    struct node *ptr;

}*top,*top1,*temp;



int count = 0;

void push(int data)
```

```
{
     if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
    printf("Node is Inserted\n\n");
}

int pop()
{
     top1 = top;
    if (top1 == NULL)
    {
        printf("\nStack Underflow\n");
        return -1;
```

```
    }
    else
        top1 = top1->ptr;
    int popped = top->info;
    free(top);
    top = top1;
    count--;
    return popped;
}

void display() {
    top1 = top;
    if (top1 == NULL)
    {
        printf("\nStack Underflow\n");
        return;
    }

    printf("The stack is \n");
    while (top1 != NULL)
    {
        printf("%d--->", top1->info);
        top1 = top1->ptr;
    }
    printf("NULL\n\n");
```

```
}
void main() {
    int choice, value;
    printf("\nImplementation of Stack using Linked List\n");
    while (1) {
        printf("\nMENU\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("\nEnter the value to insert: ");
            scanf("%d", &value);
            push(value);
            break;
        case 2:
            printf("Popped element is :%d\n", pop());
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("\Invalid Choice\n");
```

```
      }
    }
}
```

OUTPUT:

```
Implementation of Stack using Linked List

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

Enter the value to insert: 4
Node is Inserted


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

Enter the value to insert: 6
Node is Inserted


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

Enter the value to insert: 5
Node is Inserted
```

```
MENU
1. Push
2. Pop
3. Display
4. Exit
Enter your choice : 2
Popped element is :5
MENU
1. Push
2. Pop
3. Display
4. Exit
Enter your choice : 3
The stack is
6--->4--->NULL


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

QUEUE IMPLEMENTATION:

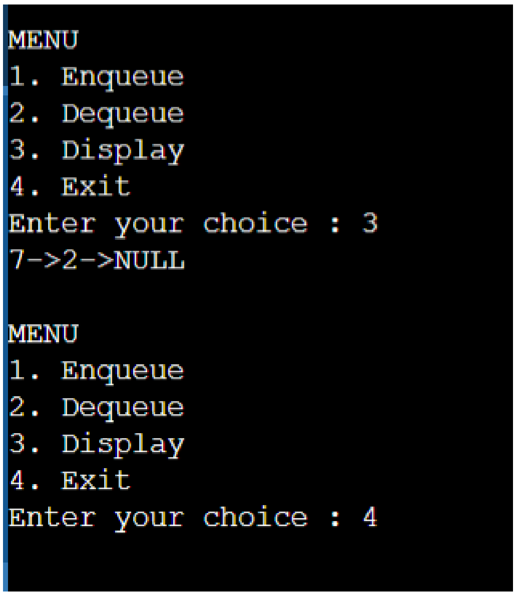
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};

struct node *front = NULL, *rear = NULL;

void enqueue(int val)
{
   struct node *newNode = malloc(sizeof(struct node));
   newNode->data = val;
   newNode->next = NULL;
   if(front == NULL && rear == NULL)
      front = rear = newNode;
   else
   {
      //add newnode in rear->next
      rear->next = newNode;
      rear = newNode;
   }
}
int dequeue()
{
   if(front == NULL)
```

```
    {
         printf("\nUNDERFLOW\n");
         return-1;
    }
    else
    {
        struct node *temp = front;
        int temp_data=front->data;
        front = front->next;
        free(temp);
        return temp_data;
    }
}
void display()
{
    struct node *temp = front;
    while(temp)
    {
        printf("%d->",temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
void main()
{
    int choice, value;
    printf("\nImplementation of Queue using Linked List\n");
    while (1) {
        printf("\nMENU\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
```

```
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
{
                case 1:
                        printf("\nEnter the value to insert: ");
                        scanf("%d", &value);
                        enqueue(value);
                        break;
                case 2:
                        printf("The element removed is :%d\n", dequeue());
                         break;
                case 3:
                         display();
                        break;
                case 4:
                        exit(0);
                        break;
                 default:
                         printf("\nInvalid Choice\n");
                }
            }
}
```

OUTPUT:

```
Implementation of Queue using Linked List

MENU
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 1

Enter the value to insert: 5

MENU
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 1

Enter the value to insert: 7

MENU
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 1

Enter the value to insert: 2

MENU
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 2
The element removed is :5
```



```
MENU
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 3
7->2->NULL

MENU
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 4
```

LAB PROGRAM 7:

Write a program 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* prev;
    struct Node* next;
};

struct Node* head = NULL;

void createlist() {
    int i, n;
    struct Node* newNode;
    struct Node* temp;
```

```
    printf("Enter the number of elements:");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter the element: ");
        scanf("%d", &newNode->data);
        if (head == NULL) {
            head = temp = newNode;
            head->prev = NULL;
            temp->next = NULL;
        } else {
            temp->next = newNode;
            newNode->prev = temp;
            temp = newNode;
            temp->next = NULL;
        }
    }
    printf("List created successfully.\n");
}

void insertLeft(struct Node* temp, int data) {
    struct Node* newNode;
    if (temp == NULL) {
        printf("Target node doesn't exist!\n");
        return;
    }
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp;
    newNode->prev = temp->prev;
    if (temp->prev != NULL) {
        temp->prev->next = newNode;
    }
    temp->prev = newNode;
    if (head == temp) {
        head = newNode;
    }
    printf("Node inserted successfully.\n");
}

void deleteNode(int key) {
    struct Node* current = head;
    while (current != NULL) {
        if (current->data == key) {
```

```
            if (current->prev != NULL) {
                current->prev->next = current->next;
            }
            if (current->next != NULL) {
                current->next->prev = current->prev;
            }
            if (current == head) {
                head = current->next;
            }
            free(current);
            printf("Node deleted successfully.\n");
            return;
        }
        current = current->next;
    }
    printf("Node with value %d not found!\n", key);
}


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

void main()
{
    int choice, data, targetValue, deleteValue;

    while(1)
   {
            printf("\nMENU:\n");
            printf("1. Create linked list\n");
            printf("2. Insert left of node\n");
            printf("3. Delete node by value\n");
             printf("4. Print the list\n");
```

```
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createlist();
                break;
            case 2:
                printf("Enter the value of the node to insert left of: ");
                scanf("%d", &targetValue);
                printf("Enter the element to insert left of the node: ");
                scanf("%d", &data);
                struct Node* temp = head;
                while (temp != NULL) {
                    if (temp->data == targetValue) {
                        insertLeft(temp, data);
                        break;
                    }
                    temp = temp->next;
                }
                break;
            case 3:
                printf("Enter the value of the node to delete: ");
                scanf("%d", &deleteValue);
                deleteNode(deleteValue);
                break;
            case 4:
                printList();
                break;
            case 5:
                exit(0);
                break;
            default:
                printf("Invalid choice.\n");
        }
    }

}
```

OUTPUT:

```
MENU:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 1
Enter the number of elements:3
Enter the element: 2
Enter the element: 4
Enter the element: 6
List created successfully.

MENU:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 2
Enter the value of the node to insert left of: 4
Enter the element to insert left of the node: 3
Node inserted successfully.
```

```
MENU:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 3
Enter the value of the node to delete: 6
Node deleted successfully.

MENU:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 4
Doubly linked list: 2 3 4

MENU:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 5
```

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>

struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};


struct TreeNode* createNode(int value) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->val = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}


struct TreeNode* insert(struct TreeNode* node, int value) {
    if (node == NULL) {
        return createNode(value);
    }
    if (value < node->val) {
        node->left = insert(node->left, value);
    } else if (value > node->val) {
        node->right = insert(node->right, value);
    }
    return node;
}

void inorderTraversal(struct TreeNode* node) {
    if (node != NULL) {
```

```
        inorderTraversal(node->left);
        printf("%d ", node->val);
        inorderTraversal(node->right);
    }
}

void postorderTraversal(struct TreeNode* node) {
    if (node != NULL) {
        postorderTraversal(node->left);
        postorderTraversal(node->right);
        printf("%d ", node->val);
    }
}

void preorderTraversal(struct TreeNode* node) {
    if (node != NULL) {
        printf("%d ", node->val);
        preorderTraversal(node->left);
        preorderTraversal(node->right);
    }
}


void main()
{
    struct TreeNode* root = NULL;
    int n, value,choice;

   printf("\nMENU:\n");
   printf("1. Create tree\n");
   printf("2. Preorder\n");
   printf("3. Inorder\n");
   printf("4. Postorder\n");
   printf("5. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);
   while(1)
  {
          switch(choice)
          {
                    case 1:
                              printf("Enter the number of nodes: ");
                              scanf("%d", &n);
```

```
                 printf("Enter the values of the nodes: ");
                for (int i = 0; i < n; ++i)
                {
                        scanf("%d", &value);
                        root = insert(root, value);
                }
                break;

        case 2:
           printf("Inorder traversal: ");
           inorderTraversal(root);
           printf("\n");

        case 3:
                printf("Inorder traversal: ");
                 inorderTraversal(root);
                printf("\n");
                break;
        case 4:
                printf("Postorder traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;

        case 5:
                exit(0);
                break;
        default:
                printf("Invalid chouce ");

        }
    }
}
```

OUTPUT:

```
MENU:
1. Create tree
2. Preorder
3. Inorder
4. Postorder
5. Exit

Enter your choice: 1
Enter the number of nodes: 6
Enter the values of the nodes: 2 5 7 8 6 9

Enter your choice: 2

Inorder traversal: 2 5 6 7 8 9

Enter your choice: 3

Inorder traversal: 2 5 6 7 8 9

Enter your choice: 4

Postorder traversal: 6 9 8 7 5 2

Enter your choice: 5
```

LAB PROGRAM 9:

a) Write a program to traverse a graph using BFS method.

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 40



struct queue {

  int items[SIZE];

  int front;

  int rear;
```

```
};

struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);

struct node {
  int vertex;
  struct node* next;
};

struct node* createNode(int);

struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};

void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();
```

```
  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
    int currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);

    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
      }
      temp = temp->next;
    }
  }
}

struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
```

```
  return newNode;
}

struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }

  return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  newNode = createNode(src);
```

```
  newNode->next = graph->adjLists[dest];

  graph->adjLists[dest] = newNode;

}


struct queue* createQueue() {

  struct queue* q = malloc(sizeof(struct queue));

  q->front = -1;

  q->rear = -1;

  return q;

}


int isEmpty(struct queue* q) {

  if (q->rear == -1)

    return 1;

  else

    return 0;

}


void enqueue(struct queue* q, int value) {

  if (q->rear == SIZE - 1)

    printf("\nQueue is Full!!");

  else {

    if (q->front == -1)

      q->front = 0;

    q->rear++;
```

```
    q->items[q->rear] = value;
  }
}

int dequeue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }
  }
  return item;
}

void printQueue(struct queue* q) {
  int i = q->front;

  if (isEmpty(q)) {
    printf("Queue is empty");
```

```
  } else {
    printf("\nQueue contains \n");
    for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}

void main() {
  int vertices, edges, src, dest;
  printf("Enter the number of vertices: ");
  scanf("%d", &vertices);
  printf("Enter the number of edges: ");
  scanf("%d", &edges);

  struct Graph* graph = createGraph(vertices);
  printf("Enter edges (source destination):\n");
  for (int i = 0; i < edges; ++i)
{
		scanf("%d%d", &src, &dest);
		addEdge(graph, src, dest);
  }
  bfs(graph, 0);
}
```

OUTPUT:

```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter edges (source destination):
0 1
0 2
1 3
2 4
Resetting queue Visited 0
Visited 2
Visited 1
Visited 4
Resetting queue Visited 3
```

b) Write a program to traverse a graph using DFS method.

```
#include <stdio.h>

#include <stdlib.h>


struct node {

  int vertex;

  struct node* next;

};


struct node* createNode(int v);


struct Graph {

  int numVertices;

  int* visited;


  struct node** adjLists;
```

```
};

void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;

  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);

  while (temp != NULL) {
    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
}

struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}
```

```
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
```

```
  graph->adjLists[dest] = newNode;
}

void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n Adjacency list of vertex %d\n ", v);
    while (temp) {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}

void main()
{
	int vertices, edges, src, dest;
	printf("Enter the number of vertices: ");
	scanf("%d", &vertices);
	printf("Enter the number of edges: ");
	scanf("%d", &edges);

	 struct Graph* graph = createGraph(vertices);
```

```
        printf("Enter edges (source destination):\n");

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

        {

                scanf("%d%d", &src, &dest);

                addEdge(graph, src, dest);

        }

        printGraph(graph);

        DFS(graph, 2);

}
```

OUTPUT:

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter edges (source destination):
0 1
0 2
1 2
2 3

 Adjacency list of vertex 0
 2 -> 1 ->

 Adjacency list of vertex 1
 2 -> 0 ->

 Adjacency list of vertex 2
 3 -> 1 -> 0 ->

 Adjacency list of vertex 3
 2 ->
Visited 2
Visited 3
Visited 1
Visited 0
```

LeetCode Programs:

1.Score of Parentheses(LP:856)

Given a balanced parentheses string s, return *the **score** of the string*.

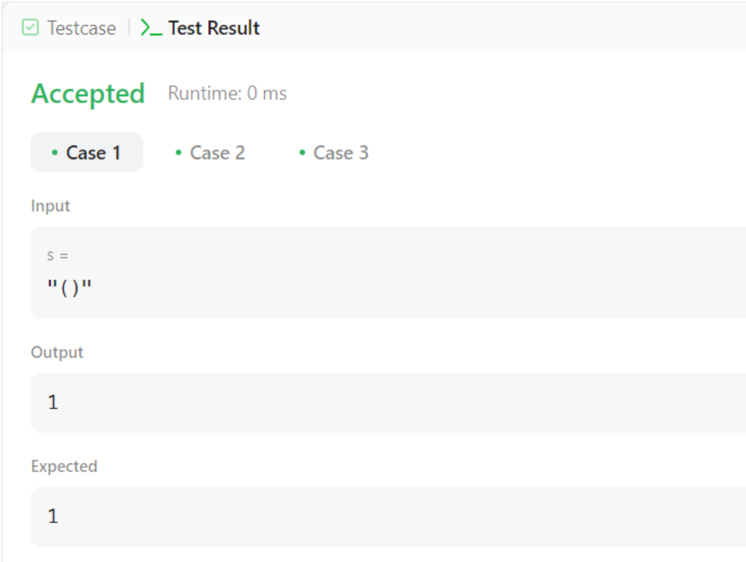
The **score** of a balanced parentheses string is based on the following rule:

- "()" has score 1.
- AB has score A + B, where A and B are balanced parentheses strings.
- (A) has score 2 * A, where A is a balanced parentheses string.

</> Code

C Auto

```c
int scoreOfParentheses(char* s)
{
    int stack[50];
    int top = -1;
    int score = 0;

    for(int i = 0; s[i] != '\0'; i++) {
        if(s[i] == '(') {
            stack[++top] = score;
            score = 0;
        } else {
            score = stack[top--] + (score == 0 ? 1 : (score * 2));
        }
    }
    return score;
}
```



2.Odd Even Linked List(LP:328)

Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Code

C Auto

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* oddEvenList(struct ListNode* head) {
    if (head == NULL || head->next == NULL || head->next->next == NULL)
        return head;

    struct ListNode *oddHead = head;
    struct ListNode *evenHead = head->next;
    struct ListNode *oddCurrent = oddHead;
    struct ListNode *evenCurrent = evenHead;

    while (evenCurrent != NULL && evenCurrent->next != NULL) {
        oddCurrent->next = evenCurrent->next;
        oddCurrent = oddCurrent->next;
        evenCurrent->next = oddCurrent->next;
        evenCurrent = evenCurrent->next;
    }
     oddCurrent->next = evenHead;

    return oddHead;
}
```

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

[1,2,3,4,5]

Output

[1,3,5,2,4]

Expected

[1,3,5,2,4]

3.Delete middle node of linked list.(LP:2095)

You are given the head of a linked list. **Delete** the **middle node**, and return *the* head *of the modified linked list*.

The **middle node** of a linked list of size n is the $\lfloor n / 2 \rfloor^{th}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x.

- For n = 1, 2, 3, 4, and 5, the middle nodes are 0, 1, 1, 2, and 2, respectively.

</> Code

C ⌄ 🔒 Auto

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
int countOfNodes(struct ListNode* head)
{
    int count = 0;
    while (head != NULL) {
        head = head->next;
        count++;
    }
    return count;
}
struct ListNode* deleteMiddle(struct ListNode* head)
{
    if (head == NULL)
        return NULL;
    if (head->next == NULL) {
        free(head);
        return NULL;
    }
    struct ListNode* copyHead = head;
    int count = countOfNodes(head);
    int mid = count / 2;
    while(mid-->1)
    head = head->next;
    head->next = head->next->next;
    return copyHead;
}
```

Testcase | Test Result

Accepted Runtime: 2 ms

• Case 1 • Case 2 • Case 3

Input

head =

```
[1,3,4,7,1,2,6]
```

Output

```
[1,3,4,1,2,6]
```

Expected

```
[1,3,4,1,2,6]
```

4.Delete a node in BST.(LP:450)

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return *the **root node reference** (possibly updated) of the BST*.

Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

</> Code

C ∨ Auto

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* minValueNode(struct TreeNode* node) {
    struct TreeNode* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL) return root;

    if (key < root->val)
        root->left = deleteNode(root->left, key);
    else if (key > root->val)
        root->right = deleteNode(root->right, key);
    else {
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }

        struct TreeNode* temp = minValueNode(root->right);
        root->val = temp->val;
        root->right = deleteNode(root->right, temp->val);
    }
    return root;
}
```

Testcase | >_ Test Result

Accepted Runtime: 2 ms

• Case 1 • Case 2 • Case 3

Input

root =
[5,3,6,2,4,null,7]

key =
3

Output

[5,4,6,2,null,null,7]

Expected

[5,4,6,2,null,null,7]

5.Bottom Left Tree Value.(LP:513)

Given the root of a binary tree, return the leftmost value in the last row of the tree.

</> Code

C Auto

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
int findBottomLeftValue(struct TreeNode* root) {
    struct TreeNode* queue[10000]; // Assuming maximum 10000 nodes
    int front = 0, rear = 0;
    queue[rear++] = root;
    int leftmostValue = root->val;

    while (front < rear) {
        int levelSize = rear - front;
        for (int i = 0; i < levelSize; i++) {
            struct TreeNode* current = queue[front++];
            if (i == 0) // first node at the current level
                leftmostValue = current->val;
            if (current->left != NULL)
                queue[rear++] = current->left;
            if (current->right != NULL)
                queue[rear++] = current->right;
        }
    }

    return leftmostValue;
}
```

Testcase | >_ Test Result

Accepted   Runtime: 5 ms

• Case 1   • Case 2

Input

root =

```
[1,2,3,4,null,5,6,null,null,7]
```

Output

```
7
```

Expected

```
7
```

Testcase | >_ Test Result

Accepted Runtime: 5 ms

• Case 1 • Case 2

Input

root =

[1,2,3,4,null,5,6,null,null,7]

Output

7

Expected

7

Testcase | >_ Test Result

Accepted Runtime: 5 ms

• Case 1 • Case 2

Input

root =

[2,1,3]

Output

1

Expected

1