VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

SWAPNIL SAHIL (1BM22CS300)

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 **SWAPNIL SAHIL (1BM22CS300)**, 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.

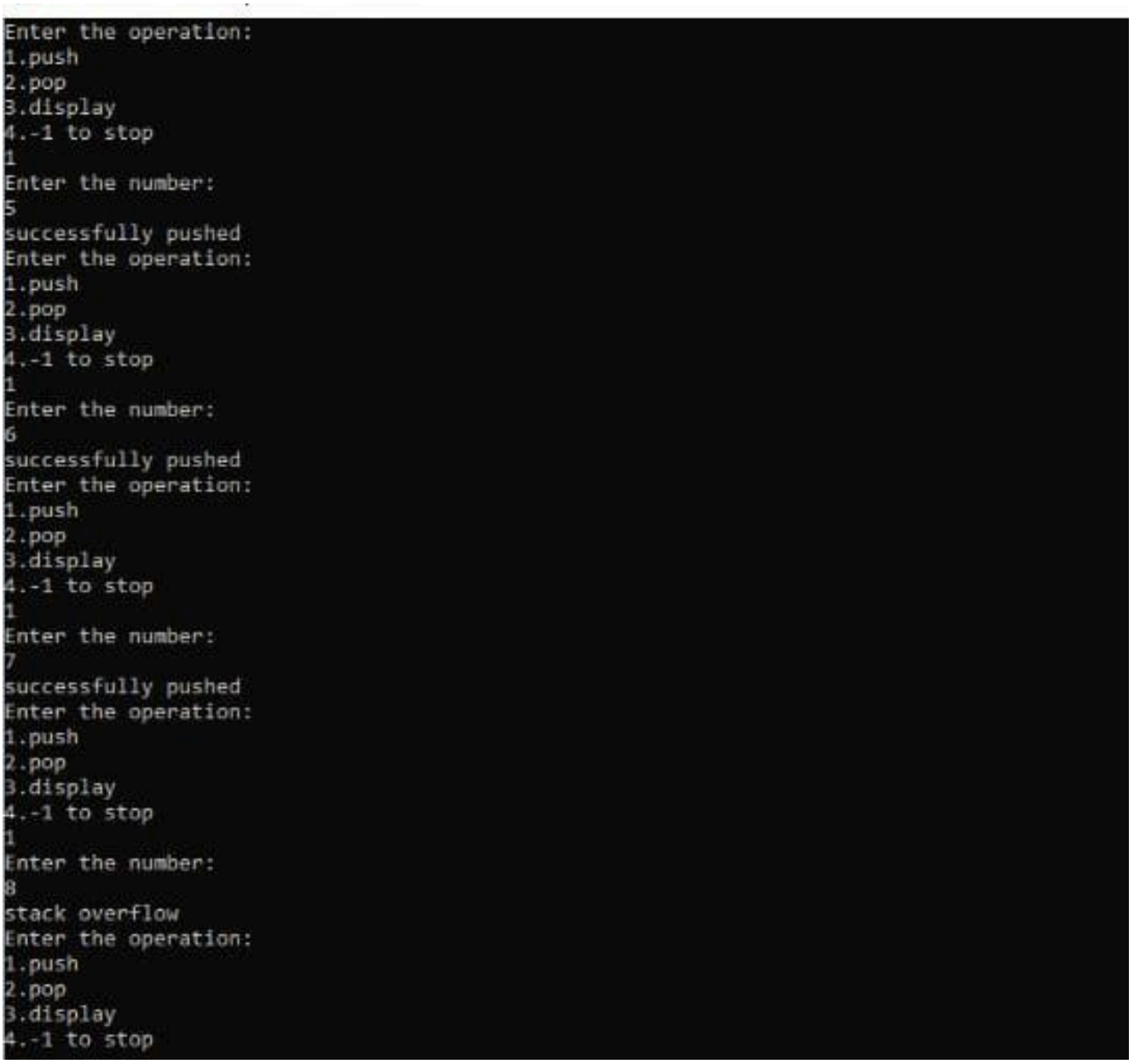
Lab program 1:

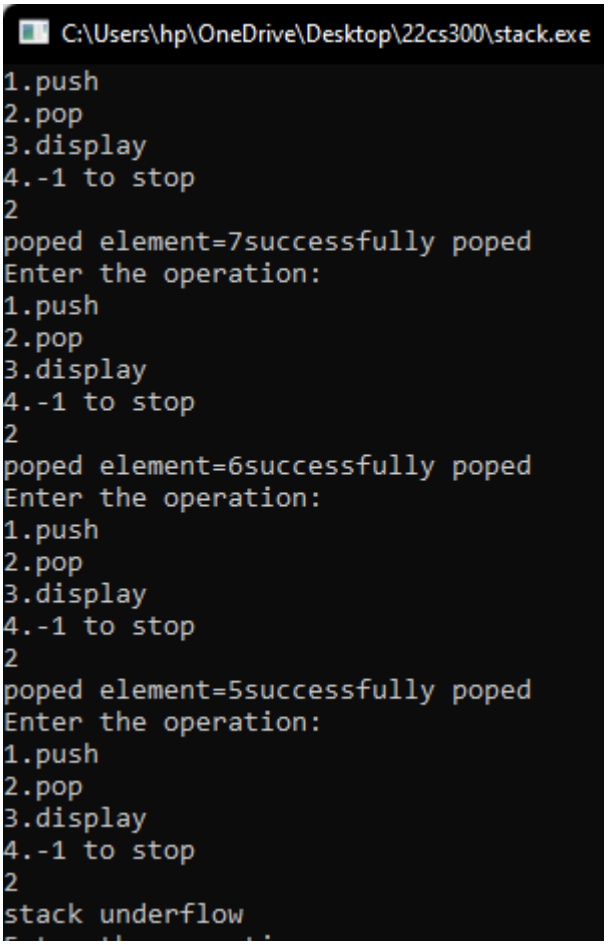
Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

```
#include<stdio.h>
#include<stdlib.h>
#define N 10

int stack[N];
int top=-1;

void push(int var);
void pop();
void display();

void main()
{
   int choice,num;

   while(1){
         printf("Enter the operation:\n1.push\n2.pop\n3.display\n4.-1 to stop\n");
         scanf("%d",&choice);
      if(choice==-1){
         printf("operation completed.\n");
         break;
      }
      else{
         switch(choice)
         {
            case 1:printf("Enter the number:\n");
```

```
                scanf("%d",&num);
                push(num);
                break;
            case 2:pop();
                break;
            case 3:display();
                break;

            default:printf("invalid input");

        }
    }
  }
}

void push(int var)
{
  if(top==N-1)
  {
    printf("stack overflow\n");
  }
  else{
    top++;
    stack[top]=var;
    printf("successfully pushed\n");
  }
}

void pop()
{
  if(top==-1)
  {
    printf("stack underflow\n");
  }
  else{
    printf("poped element=%d",stack[top]);
    printf("successfully poped\n");
    top--;
  }
}

void display()
{
  int i;
  for(i=top;i>=0;i--)
  {
    printf("%d",stack[i]);
  }
}
```

Output:


Enter the operation:
1.push
2.pop
3.display
4.-1 to stop
1
Enter the number:
5
successfully pushed
Enter the operation:
1.push
2.pop
3.display
4.-1 to stop
1
Enter the number:
6
successfully pushed
Enter the operation:
1.push
2.pop
3.display
4.-1 to stop
1
Enter the number:
7
successfully pushed
Enter the operation:
1.push
2.pop
3.display
4.-1 to stop
1
Enter the number:
8
stack overflow
Enter the operation:
1.push
2.pop
3.display
4.-1 to stop


C:\Users\hp\OneDrive\Desktop\22cs300\stack.exe
1.push
2.pop
3.display
4.-1 to stop
2
poped element=7successfully poped
Enter the operation:
1.push
2.pop
3.display
4.-1 to stop
2
poped element=6successfully poped
Enter the operation:
1.push
2.pop
3.display
4.-1 to stop
2
poped element=5successfully poped
Enter the operation:
1.push
2.pop
3.display
4.-1 to stop
2
stack underflow

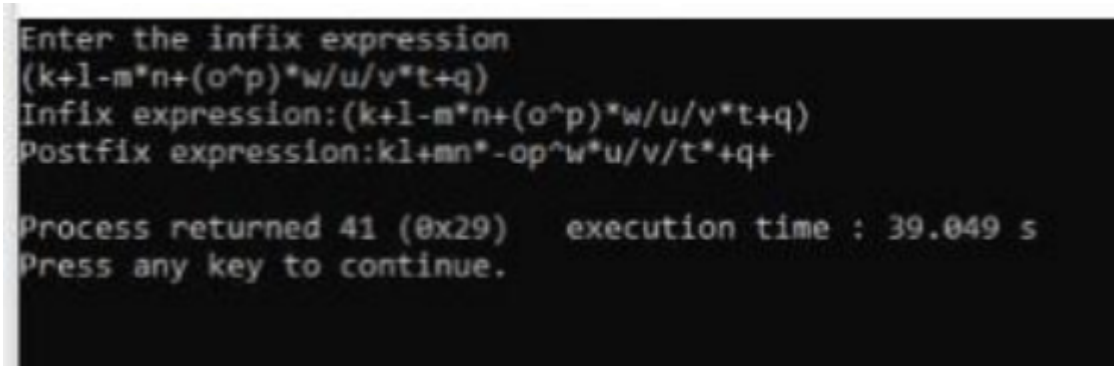
Lab program 2:

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>
#include<string.h>
int index1=0,pos=0,top=-1,length;
char symbol,temp,infix[50],postfix[50],stack[50];
void infixtopostfix();
void push(char symbol);
char pop();
int pred(char symb);
void main(){
  printf("Enter the infix expression\n");
  scanf("%s",infix);
  infixtopostfix();
  printf("Infix expression:%s\n",infix);
  printf("Postfix expression:%s\n",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 '/':
      case '^':while(pred(stack[top])>=pred(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++;
  stack[top]=symbol;
}
char pop(){
  char symb;
  symb=stack[top];
  top--;
  return(symb);
}
int pred(char symbol){
  int p;
  switch(symbol){
    case '^':p=3;
      break;
    case '*':
    case '/':p=2;
     break;
    case '+':
    case '-':p=1;
     break;
    case '(':p=0;
     break;
    case '#':p=-1;
     break;
  }
  return(p);
}
```

OUTPUT:



Lab program 3a:

WAP to simulate the working of a queue of integers using an array. Provide the following operations

a) Insert

b) Delete

c) Display

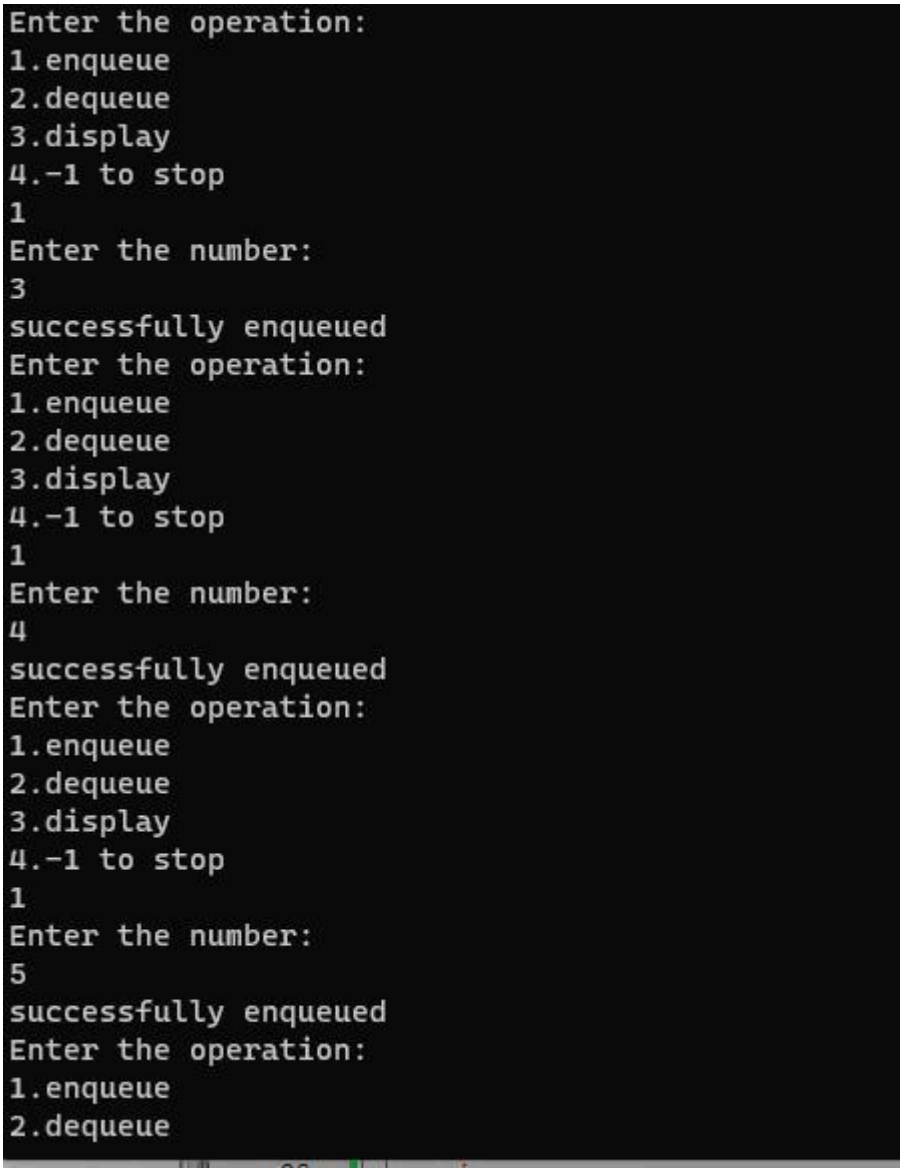
The program should print appropriate messages for queue empty and queue overflow conditions.

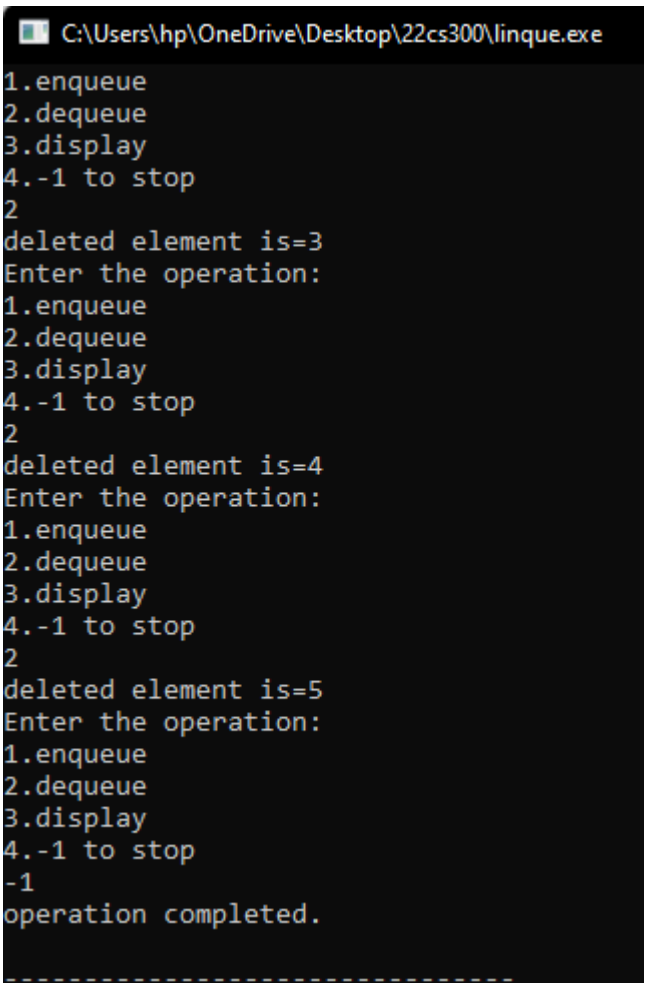
```
#include<stdio.h>
#define N 3

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

void enqueue(int x)
{
	if(rear==N-1)
	{
		printf("queue overflow\n");
	}
	else if(front==-1 && rear==-1)
	{
		front=rear=0;
		queue[rear]=x;
		printf("successfully enqueued\n");
	}
	else
	{
		rear++;
		queue[rear]=x;
		printf("successfully enqueued\n");
	}
}

void dequeue()
{
	if(front==-1)
	{
		printf("queue underflow\n");
	}
	else if(front==rear)
	{
		printf("deleted element is=%d\n",queue[front]);
		front=rear=-1;
	}
```

```
        else
        {
                printf("deleted element is=%d\n",queue[front]);
                front++;
        }

}
void display()
{
        if(front==-1)
        {
                printf("queue is empty\n");
        }
        else
        {
                int i;
                printf("Elements are:\n");
                for(i=front;i<=rear;i++)
                {
                        printf("%d\n",queue[i]);
                }
        }

}

void main()
{
   int choice,num;

   while(1){
        printf("Enter the operation:\n1.enqueue\n2.dequeue\n3.display\n4.-1 to stop\n");
        scanf("%d",&choice);
      if(choice==-1){
        printf("operation completed.\n");
        break;
      }
      else{
        switch(choice)
        {
          case 1:printf("Enter the number:\n");
                scanf("%d",&num);
                enqueue(num);
                break;
          case 2:dequeue();
                break;
          case 3:display();
                break;

          default:printf("invalid input");
```

```
        }
      }
    }
}
```

OUTPUT:


Enter the operation:
1.enqueue
2.dequeue
3.display
4.-1 to stop
1
Enter the number:
3
successfully enqueued
Enter the operation:
1.enqueue
2.dequeue
3.display
4.-1 to stop
1
Enter the number:
4
successfully enqueued
Enter the operation:
1.enqueue
2.dequeue
3.display
4.-1 to stop
1
Enter the number:
5
successfully enqueued
Enter the operation:
1.enqueue
2.dequeue



Lab program 3b:

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations.

a) Insert

b) Delete

c) Display

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

```
#include<stdio.h>

#include<stdlib.h>

#define N 3
```

```
int queue[N],front=-1,rear=-1;

void enqueue(int x)
{
	if((rear+1)%N==front)
	{
		printf("queue overflow\n");
	}
	else if(front==-1 && rear==-1)
	{
		front=rear=0;
		queue[rear]=x;
		printf("successfully enqueued\n");
	}
	else
	{
		rear=(rear+1)%N;
		queue[rear]=x;
		printf("successfully enqueued\n");
	}
}

void dequeue()
{
	if(front==-1)
	{
		printf("queue underflow\n");
```

```
        }
        else if(front==rear)
        {
                printf("deleted element is=%d\n",queue[front]);
                front=rear=-1;
        }
        else
        {
                printf("deleted element is=%d\n",queue[front]);
                front=(front+1)%N;
        }

}
void display()
{
        if(front==-1)
        {
                printf("queue is empty\n");
        }
        else
        {
                int i;
                printf("Elements are:\n");
                for(i=front;i!=rear;i=(i+1)%N)
                {
                        printf("%d\n",queue[i]);
                }
```

```
                printf("%d\n",queue[i]);
        }

}

void main()
{
   int choice,num;

   while(1){
        printf("Enter the operation:\n1.enqueue\n2.dequeue\n3.display\n4.-1 to stop\n");
        scanf("%d",&choice);
      if(choice==-1){
        printf("operation completed.\n");
        break;
      }
      else{
        switch(choice)
        {
          case 1:printf("Enter the number:\n");
                scanf("%d",&num);
                enqueue(num);
                break;
          case 2:dequeue();
                break;
          case 3:display();
                break;
```

```
        default:printf("invalid input");


            }
        }
    }
}
```

OUTPUT:

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 2
Inserted 2 into the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 3
Inserted 3 into the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 4
Inserted 4 into the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 5
Inserted 5 into the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 6
Inserted 6 into the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 7
Queue Overflow! Cannot insert element.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 2 3 4 5 6

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 2 from the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 3 from the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 4 from the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 5 from the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 6 from the queue.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow! Cannot delete element.

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

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.

c) Display the contents of the linked list.

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

struct node{
        int data;
        struct node * next;
};
struct node *head=0,*newnode,*temp;
void create()
{
        int i,n;
        printf("Enter the no. of elements:\n");
        scanf("%d",&n);

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

           if(head==0)
```

```
            {
                temp=head=newnode;
                }
                else
                {
                        temp->next=newnode;
                        temp=newnode;
                }
        }
}
void display()
{
   temp=head;
        printf("The elements are:\n");
        while(temp!=0)
        {
                printf("%d\n",temp->data);
                temp=temp->next;
                length++;
        }
}
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=0;
        temp=head;
        while(temp->next!=0)
        {
                temp=temp->next;
        }
        temp->next=newnode;
}
void insert_loc()
{
        int pos,i=0;
        newnode=(struct node *)malloc(sizeof(struct node));
        printf("Enter the position:\n");
        scanf("%d",&pos);


        if(pos<0)
        {
                printf("invalid position\n");
        }
```

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


        }
}
void main()
{
        int choice;
        while(1)
        {
                printf("Enter operation:\n1.create\n2.display\n3.insert at beginning\n4.insert at
end\n5.insert after a location\n.-1 to end\n");
                scanf("%d",&choice);


                if(choice==-1)
                {
                        printf("operation completed!\n");
```

```
                break;
        }
        else
        {
                switch(choice)
                {
                        case 1:create();
                                break;
                   case 2:display();
                          break;
                   case 3:insert_beg();
                                break;
                        case 4:insert_end();
                                break;
                        case 5:insert_loc();
                                break;
                        default:printf("invalid input\n");
                }
        }
    }


}
```

OUTPUT:

```
Enter operation:
1.create
2.display
3.insert at beginnning
4.insert at end
5.insert at position
6.-1 to end
1
enter the number of elements:
2
Enter the element 1:
3
Enter the element 2:
4
Enter operation:
1.create
2.display
3.insert at beginnning
4.insert at end
5.insert at position
6.-1 to end
2
3
4
Enter operation:
1.create
2.display
3.insert at beginnning
4.insert at end
5.insert at position
6.-1 to end
3
Enter the new element:
5
Enter operation:
1.create
2.display
3.insert at beginnning
4.insert at end
5.insert at position
6.-1 to end
2
5
3
4
Enter operation:
1.create
2.display
3.insert at beginnning
4.insert at end
5.insert at position
```

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.

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

struct node{
        int data;
        struct node * next;
};
struct node *head=0,*newnode,*temp;
void create()
{
        int i,n;
        printf("Enter the no. of elements:\n");
        scanf("%d",&n);

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

           if(head==0)
           {
                temp=head=newnode;
```

```
                }
                else
                {
                        temp->next=newnode;
                        temp=newnode;
                }
        }
}
void display()
{
   temp=head;
        printf("The elements are:\n");
        while(temp!=0)
        {
                printf("%d\n",temp->data);
                temp=temp->next;
                length++;
        }
}
void delete_beg()
{
        temp=head;
        if(head==0)
        {
                printf("empty");
        }
        else
        {
                head=temp->next;
```

```
                free(temp);
        }

}
void delete_end()
{
        temp=head;
        struct node *prenode;
        while(temp->next!=0)
        {
                prenode=temp;
                temp=temp->next;
        }
        if(temp=head)
        {
                head=0;
        }
        else
        {
                prenode->next=0;
        }
        free(temp);
}
void delete_pos()
{
        struct node *nextnode;
        int pos,i=1;
        temp=head;
        printf("enter position\n");
```

```
        scanf("%d",&pos);
        while(i<pos)
        {
                temp=temp->next;
                i++;
        }
        nextnode=temp->next;
        temp->next=nextnode->next;
        free(nextnode);
}

void main()
{
        int choice;
        while(1)
        {
                printf("Enter operation:\n1.create\n2.display\n3.delete at beginning\n4.delete at end\n5.delete at a location\n.-1 to end\n");
                scanf("%d",&choice);

                if(choice==-1)
                {
                        printf("operation completed!\n");
                        break;
                }
                else
                {
                        switch(choice)
                        {
                                case 1:create();
```

```
                    break;
                case 2:display();
                    break;
                case 3:delete_beg();
                    break;
                case 4:delete_end();
                    break;
                case 5:delete_loc();
                    break;
                default:printf("invalid input\n");
            }
        }
    }

}
```

OUTPUT:

```
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end
1
enter the number of elements:
4
Enter the element 1:
2
Enter the element 2:
3
Enter the element 3:
4
Enter the element 4:
5
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end
2
The elements are:
2
3
4
5
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end
3
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end
2
The elements are:
3
4
5
```

Lab program 6a:

WAP to Implement Single Link List with following operations

a) Sort the linked list.

b) Reverse the linked list.

c) Concatenation of two linked lists

```
#include<stdio.h>

struct node{
        int data;
        struct node * next;
};
struct node *head=0,*newnode,*temp;
int length=0;
void create()
{
        int i,n;
        printf("Enter the no. of elements:\n");
        scanf("%d",&n);

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

           if(head==0)
           {
                temp=head=newnode;
```

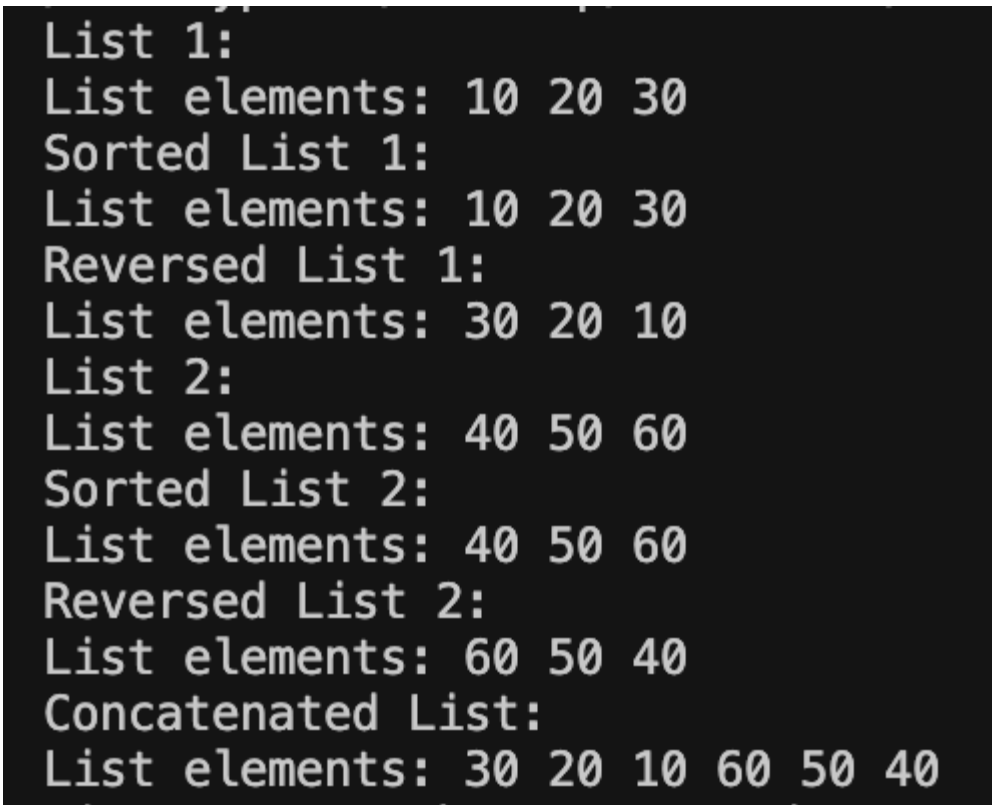
```
                }
                else
                {
                        temp->next=newnode;
                        temp=newnode;
                }
        }
}
void display()
{
   temp=head;
        printf("The elements are:\n");
        while(temp!=0)
        {
                printf("%d\n",temp->data);
                temp=temp->next;
                length++;
        }
}

void reverse()
{
        struct node *prenode=0,*curnode=head,*nextnode=head;

        while(nextnode != 0)
        {
                nextnode = nextnode->next;
                curnode->next=prenode;
                prenode=curnode;
```

```
                curnode=nextnode;
        }
        head=prenode;
}

void sort()
{
        int swapped=0;
        struct node *end=0;
        if (head == NULL || head->next == NULL) {
      printf("already sorted"); // Already sorted or empty list
   }
        do{
           swapped=0;
                temp=head;
                while(temp->next != end)
                {

                if(temp->data > temp->next->data)
                {
                        int t =temp->data;
                        temp->data=temp->next->data;
                        temp->next->data=t;
                        swapped=1;
                }
//              printf("%d\n",temp->data);
                temp=temp->next;
           }
           end=temp;
```

```
    } while(swapped);



}



void main()
{
        int choice;
        printf("Enter operation:\n1.create\n2.display\n3.reverse\n4.sort\n5.concat\n.-1 to
end\n");
        while(1)
        {
                printf("Enter operation:\n");
                scanf("%d",&choice);


                if(choice==-1)
                {
                        printf("operation completed!\n");
                        break;
                }
                else
                {
                        switch(choice)
                        {
                                case 1:create();
                                        break;
                           case 2:display();
                                    break;
```

```
                        case 3:reverse();
                                break;
                         case 4:sort();
                                break;
                        case 5:printf("Enter first list\n");
                              create();
                              printf("enter second list\n");
                              create();
                              break;
                               default:printf("invalid input\n");
                      }
                }
        }


}
```

OUTPUT:



Lab program 6b:

WAP to implement Stack & Queues using Linked Representation

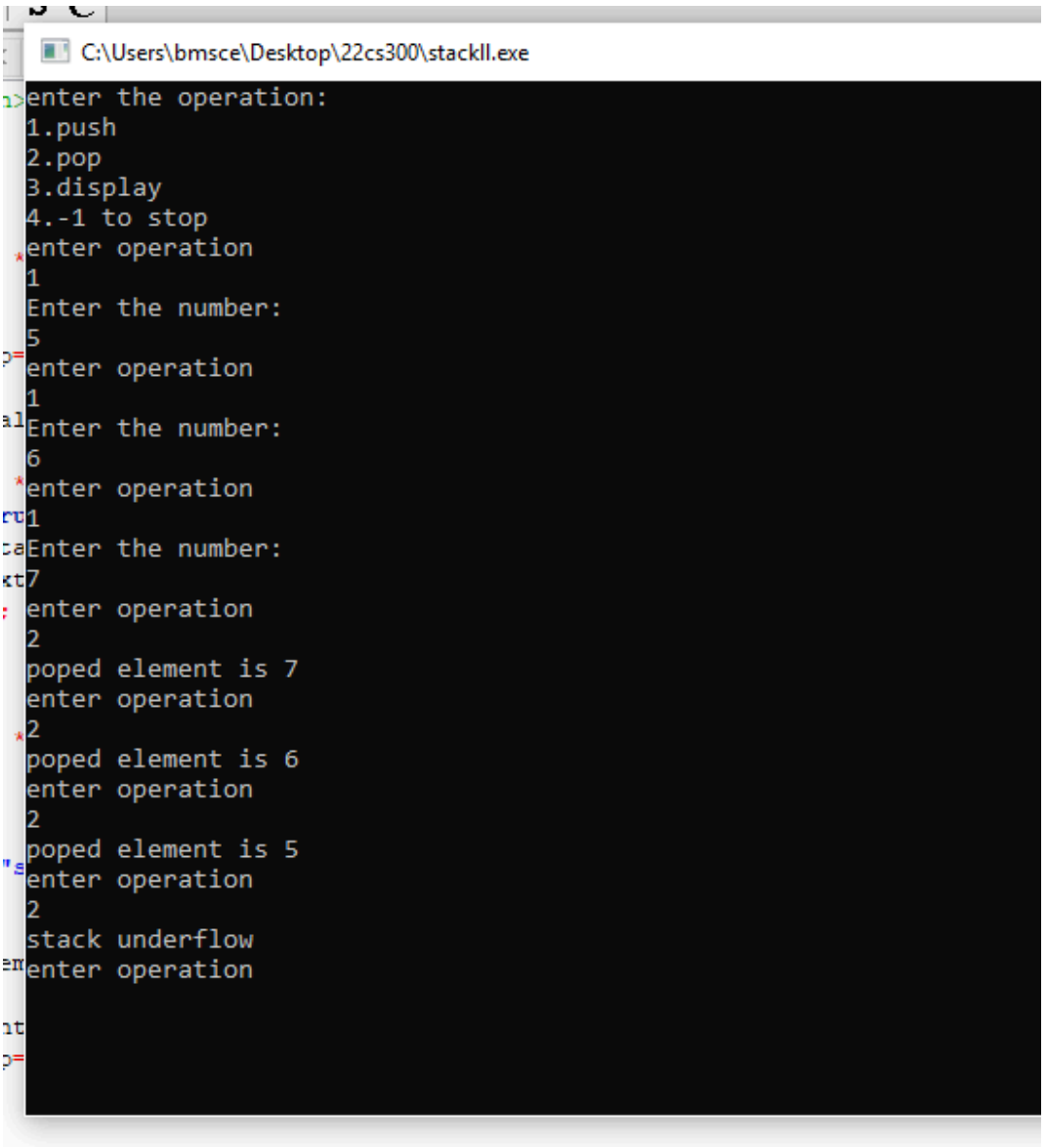
```
#include<stdio.h>
#include<stdlib.h>

struct node
{
        int data;
        struct node * link;

};

struct node *top=0;

void push(int val)
```

```
{
        struct node * newnode;
        newnode=(struct node *)malloc(sizeof(struct node));
        newnode->data=val;
        newnode->link=top;
        top=newnode;
}

void display()
{
        struct node *temp;
        temp=top;
        if(top==0)
        {
                printf("empty");
        }
        else
        {
                while(temp!=0)
                {
                        printf("%d\n",temp->data);
                        temp=temp->link;
                }
        }
}

void pop()
{
        struct node *temp;
```

```
        temp=top;
        if(top==0)
        {
                printf("empty");
        }
        else
        {
                printf("poped is %d",top->data);
                top=top->link;
                free(temp);
        }
}

void main()
{
         int choice,num;

   while(1){
        printf("Enter the operation:\n1.push\n2.pop\n3.display\n4.-1 to stop\n");
        scanf("%d",&choice);
      if(choice==-1){
        printf("operation completed.\n");
        break;
      }
      else{
        switch(choice)
        {
           case 1:printf("Enter the number:\n");
                scanf("%d",&num);
```

```
                push(num);
                break;
            case 2:pop();
                break;
            case 3:display();
                break;

            default:printf("invalid input");

        }
    }
  }
}
```

OUTPUT:



QUEUE:

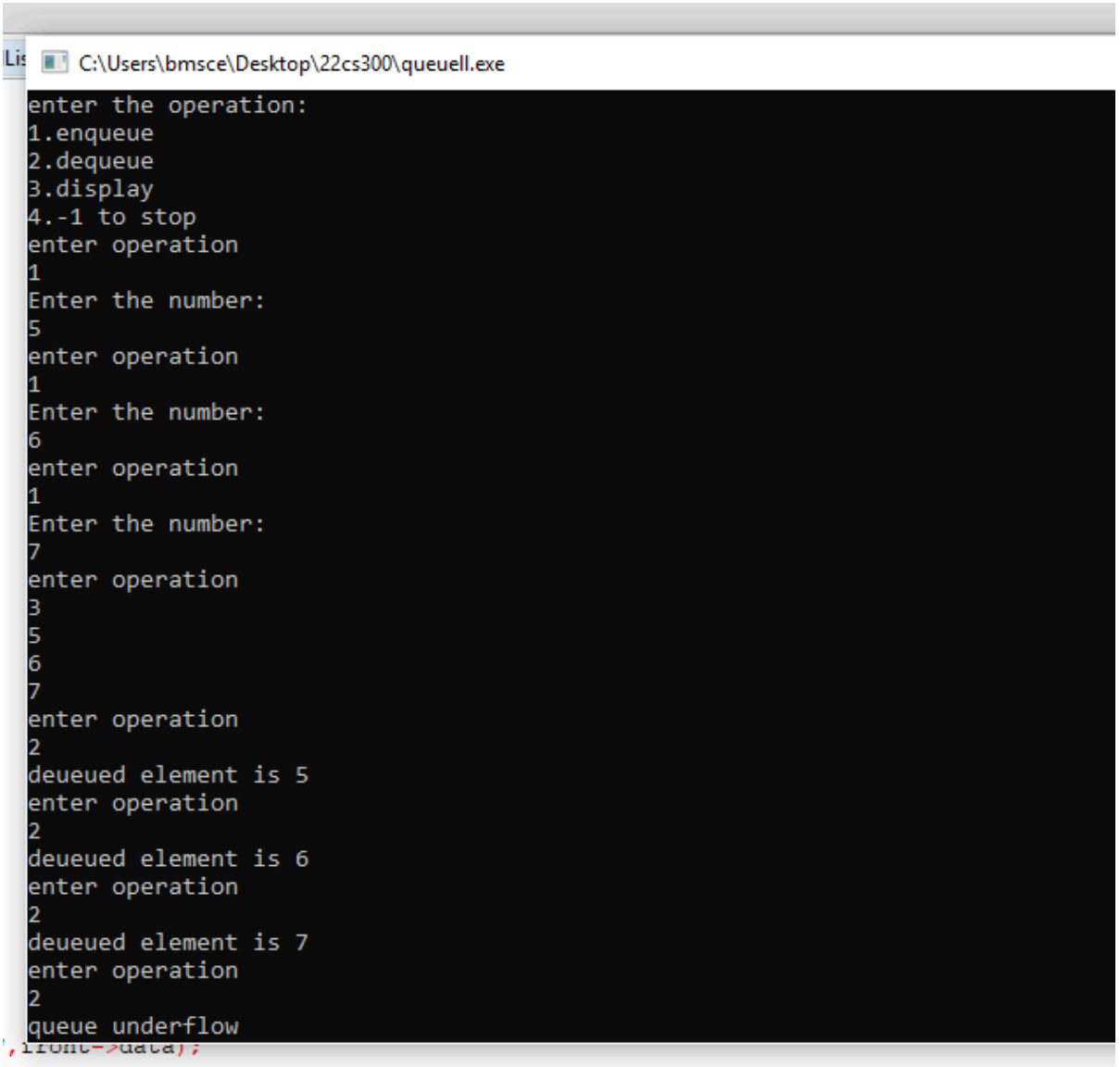
```
#include<stdio.h>
#include<stdlib.h>

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

```
void display(struct Node* front) {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }

    struct Node* temp = front;
    printf("Queue elements are:\t");
    while (temp != NULL) {
        printf("%d\t", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Queue Overflow\n");
        return;
    }

    newNode->data = data;
    newNode->next = NULL;

    if (*rear == NULL) {
        *front = *rear = newNode;
        return;
```

```
    }

    (*rear)->next = newNode;
    *rear = newNode;
}

int dequeue(struct Node** front, struct Node** rear) {
    if (*front == NULL) {
        printf("Queue Underflow\n");
        return -1; // You can choose another value to indicate underflow
    }

    struct Node* temp = *front;
    int dequeuedData = temp->data;

    *front = (*front)->next;

    if (*front == NULL) {
        *rear = NULL; // If the last element is dequeued, update rear
    }

    free(temp);
    return dequeuedData;
}

int main() {
    int op, n, dequeuedElement;
    struct Node* front = NULL;
    struct Node* rear = NULL;
```

```
    printf("Enter 1. Enqueue\n2. Dequeue\n3. -1 to stop\n");
    while (1) {
        printf("Enter operation\n");
        scanf("%d", &op);

        if (op == -1) {
            printf("Execution stopped\n");
            break;
        }

        switch (op) {
            case 1:
                printf("Enter the element to enqueue\n");
                scanf("%d", &n);
                enqueue(&front, &rear, n);
                break;
            case 2:
                dequeuedElement = dequeue(&front, &rear);
                if (dequeuedElement != -1) {
                    printf("Dequeued Element: %d\n", dequeuedElement);
                }
                break;
        }
        display(front);
    }

    return 0;
}
```

OUTPUT:


C:\Users\bmsce\Desktop\22cs300\queuell.exe
enter the operation:
1.enqueue
2.dequeue
3.display
4.-1 to stop
enter operation
1
Enter the number:
5
enter operation
1
Enter the number:
6
enter operation
1
Enter the number:
7
enter operation
3
5
6
7
enter operation
2
deueued element is 5
enter operation
2
deueued element is 6
enter operation
2
deueued element is 7
enter operation
2
queue underflow

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>

struct node{
        int data;
        struct node *next;
        struct node *prev;
};

struct node *head=0,*newnode,*temp;

void create()
{
        int i,n;
        printf("Enter the no. of elements:\n");
        scanf("%d",&n);

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

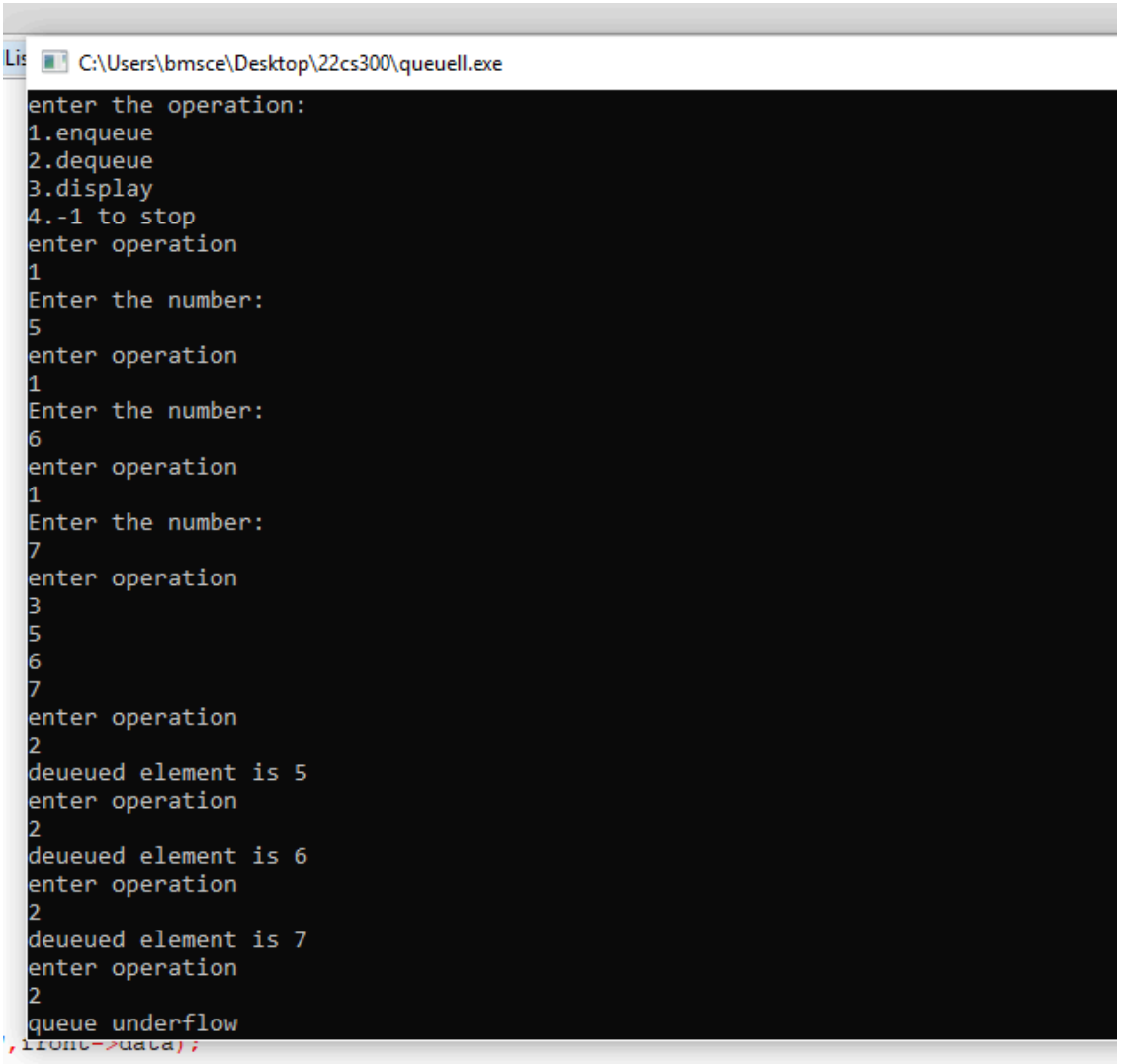
```
            if(head==0)
            {
                  temp=head=newnode;
                  }
                  else
                  {
                        temp->next=newnode;
                        newnode->prev=temp;
                        temp=newnode;
                  }
      }
}

void display()
{
      temp=head;
      while(temp!=0)
      {
            printf("%d\n",temp->data);
            temp=temp->next;
      }
}
void insert_left(){
int node,i=1;
printf("enter data\n");
scanf("%d",node);
temp=head;
if(node<1)
```

```
{
printf(“invalid position”);}
else if(node==1)
{
        newnode=(struct node *)malloc(sizeof(struct node));
        printf("enter data\n");
        scanf("%d",newnode->data);
        newnode->prev=0;
        head->prev=newnode;
        newnode->next=head;
        head=newnode;
}
else
{
newnode=(struct node *)malloc(sizeof(struct node));
printf("enter data\n");
scanf("%d",newnode->data);
while(i<node-1)
{
temp=temp->next;
i++;
}
newnode->prev=temp;
newnode->next=temp->next;
temp->next=newnode;
newnode->next->prev=newnode;
}
}
void delete_pos()
```

```
{
        int pos,i=1;
        temp=head;
        printf("enter position\n");
        scanf("%d",&pos);
        while(i<pos)
        {
                temp=temp->next;
                i++;
        }
        temp->prev->next=temp->next;
        temp->next->prev=temp->prev;
       free(temp);
}
void main()
{
        int choice;
        while(1)
        {
                printf("Enter operation:\n1.create\n2.display\n3.insert at beginning\n4.insert at
end\n5.insert after a location\n.-1 to end\n");
                scanf("%d",&choice);

                if(choice==-1)
                {
                        printf("operation completed!\n");
                        break;
                }
                else
                {
```

```
                    switch(choice)
                    {
                            case 1:create();
                                    break;
                       case 2:display();
                              break;
                       case 3:insert_left();
                                    break;
                              case 4:delete_pos();
                                    break;
                              default:printf("invalid input\n");
                    }
              }
        }


}
```

OUTPUT:

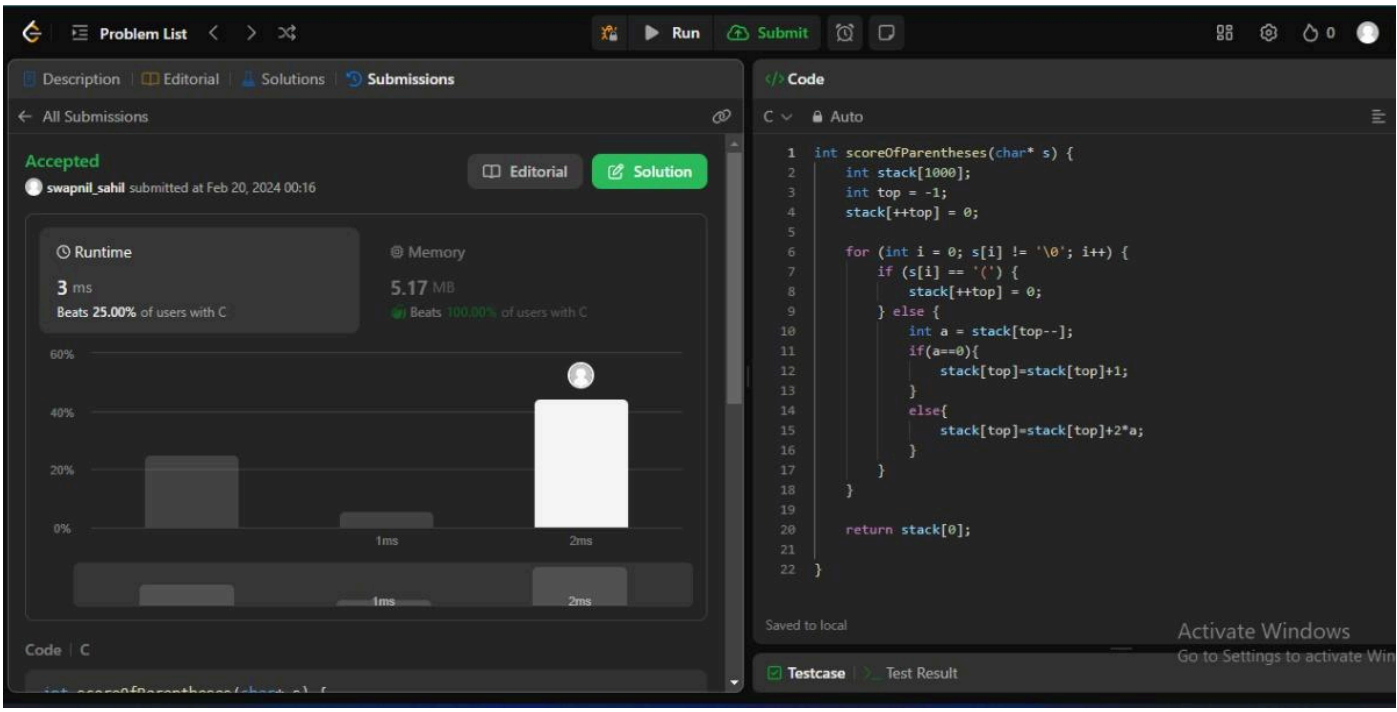


LeetCode Problem:

ScoreOfParentheses:

```
int scoreOfParentheses(char* s) {

    int stack[1000];

    int top = -1;

    stack[++top] = 0;



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

        if (s[i] == '(') {

            stack[++top] = 0;

        } else {
```

```
            int a = stack[top--];
            if(a==0){
                stack[top]=stack[top]+1;
            }
            else{
                stack[top]=stack[top]+2*a;
            }
        }
    }

    return stack[0];

}
```



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 andpost order

c) To display the elements in the tree.

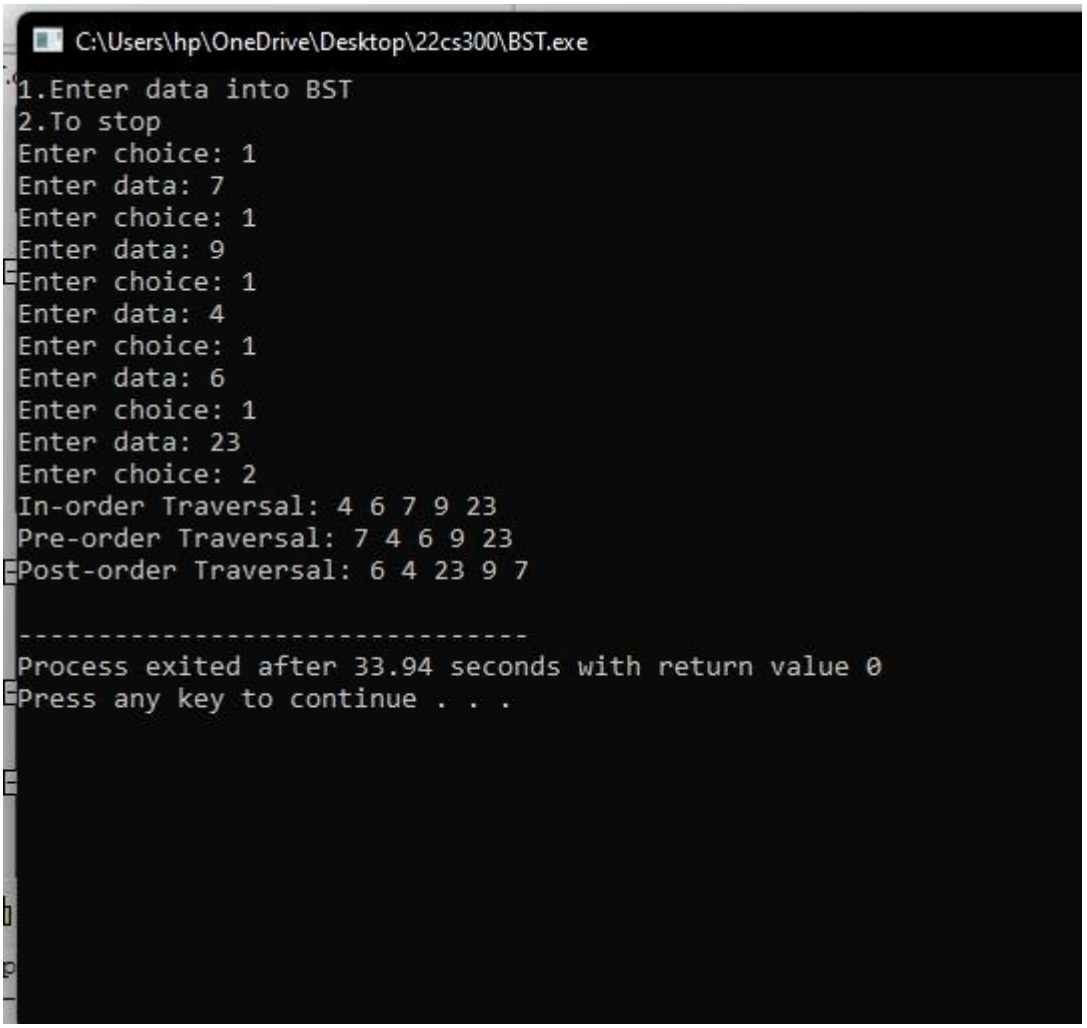
```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
```

```
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

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

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

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

```
void display(struct Node* root) {
    printf("In-order Traversal: ");
    inOrder(root);
    printf("\nPre-order Traversal: ");
    preOrder(root);
    printf("\nPost-order Traversal: ");
    postOrder(root);
    printf("\n");
}

int main() {
    struct Node* root = NULL;
    int data,c;
    printf("1.Enter data into BST\n2.To stop\n");
    while(1){
      printf("Enter choice: ");
      scanf("%d",&c);
      switch(c){
        case 1:
        printf("Enter data: ");
        scanf("%d", &data);
        root = insert(root, data);
        break;
        case 2:
        display(root);
        exit(0);
      }
    }
```

```
    return 0;
}
```

OUTPUT:



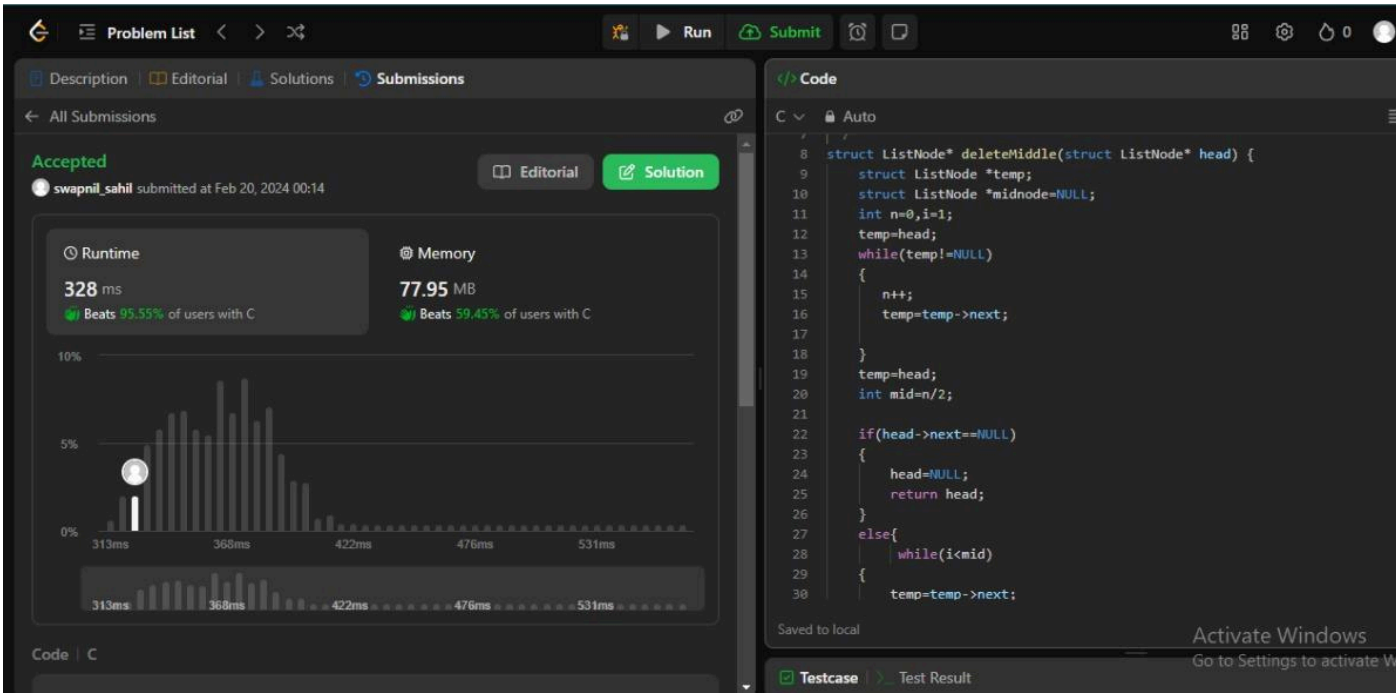
Leet Code Problem:

Delete the Middle Node Of a Linked List:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
```

```
 * };
 */
struct ListNode* deleteMiddle(struct ListNode* head) {
    struct ListNode *temp;
    struct ListNode *midnode=NULL;
    int n=0,i=1;
    temp=head;
    while(temp!=NULL)
    {
       n++;
       temp=temp->next;

    }
    temp=head;
    int mid=n/2;

    if(head->next==NULL)
    {
        head=NULL;
        return head;
    }
    else{
         while(i<mid)
    {
        temp=temp->next;
        i++;
    }
    }
```

```
    midnode=temp->next;
    temp->next=midnode->next;
    free(midnode);

    return head;


}
```

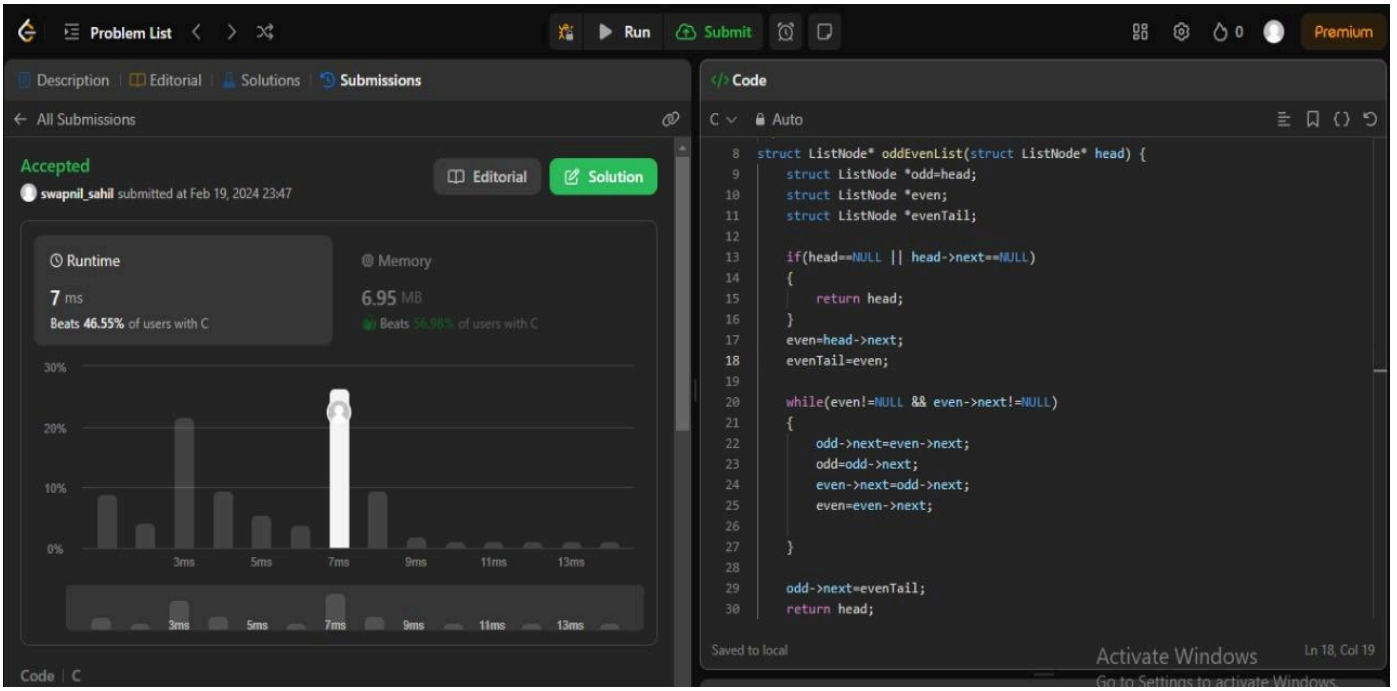


Leet Code Problem:

Odd Even Linked List:

```
struct ListNode* oddEvenList(struct ListNode* head) {

    struct ListNode *odd=head;

    struct ListNode *even;
```

```
    struct ListNode *evenTail;



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

    {

        return head;

    }

    even=head->next;

    evenTail=even;



    while(even!=NULL && even->next!=NULL)

    {

        odd->next=even->next;

        odd=odd->next;

        even->next=odd->next;

        even=even->next;



    }



    odd->next=evenTail;

    return head;

}
```


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
All Submissions
Accepted
swapnil_sahil submitted at Feb 19, 2024 23:47
Editorial
Solution
Runtime
7 ms
Beats 46.55% of users with C
Memory
6.95 MB
Beats 56.98% of users with C
30%
20%
10%
0%
3ms
5ms
7ms
9ms
11ms
13ms
Code | C
Code
C
Auto
struct ListNode* oddEvenList(struct ListNode* head) {
struct ListNode *odd=head;
struct ListNode *even;
struct ListNode *evenTail;
if(head==NULL || head->next==NULL)
{
return head;
}
even=head->next;
evenTail=even;
while(even!=NULL && even->next!=NULL)
{
odd->next=even->next;
odd=odd->next;
even->next=odd->next;
even=even->next;
}
odd->next=evenTail;
return head;
Saved to local
Ln 18, Col 19
Activate Windows

Lab program 9:

Write a Program to traverse a graph using BFS method.

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

#define MAX_NODES 100

// Define a structure for a node in the graph
struct Node {
    int data;
    struct Node* next;
};

// Define a structure for the graph
struct Graph {
    int numNodes;
    struct Node* adjLists[MAX_NODES];
    int visited[MAX_NODES];
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

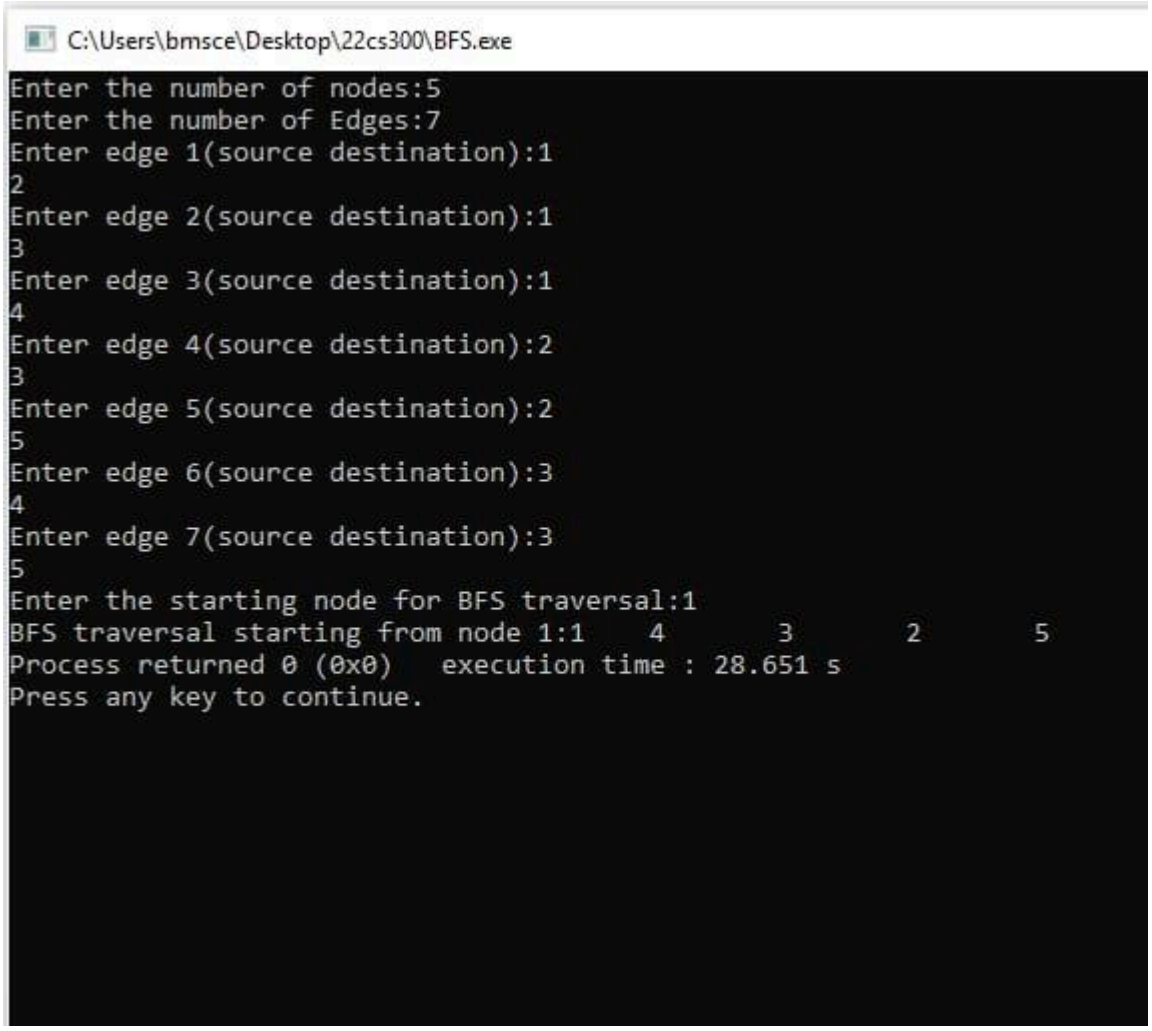
```
// Function to create a graph with n nodes
struct Graph* createGraph(int n) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numNodes = n;
    for (int i = 0; i < n; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to 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;
}

// Function to perform Breadth First Search
void BFS(struct Graph* graph, int startNode) {
```

```
    // Create a queue for BFS
    int queue[MAX_NODES];
    int front = 0, rear = 0;

    // Mark the current node as visited and enqueue it
    graph->visited[startNode] = 1;
    queue[rear++] = startNode;

    while (front < rear) {
        // Dequeue a vertex from queue and print it
        int current = queue[front++];
        printf("%d ", current);

        // Get all adjacent vertices of the dequeued vertex current
        // If an adjacent has not been visited, then mark it visited and enqueue it
        struct Node* temp = graph->adjLists[current];
        while (temp) {
            int adjNode = temp->data;
            if (!graph->visited[adjNode]) {
                graph->visited[adjNode] = 1;
                queue[rear++] = adjNode;
            }
            temp = temp->next;
        }
    }
}
```

```
int main() {
    // Get the number of nodes from the user
    int numNodes;
    printf("Enter the number of nodes: ");
    scanf("%d", &numNodes);

    // Create a graph with the specified number of nodes
    struct Graph* graph = createGraph(numNodes);

    // Get the number of edges from the user
    int numEdges;
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    // Add edges
    for (int i = 0; i < numEdges; i++) {
        int src, dest;
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    // Print BFS traversal
    int startNode;
    printf("Enter the starting node for BFS traversal: ");
    scanf("%d", &startNode);
    printf("BFS traversal starting from node %d: ", startNode);
```

```
    BFS(graph, startNode);

    return 0;
}
```

OUTPUT:



b)Write a program to check wheater given graph is connected or not using DFS method

```
#include <stdio.h>

#include <stdlib.h>


#define MAX_NODES 100


// Define a structure for a node in the graph
```

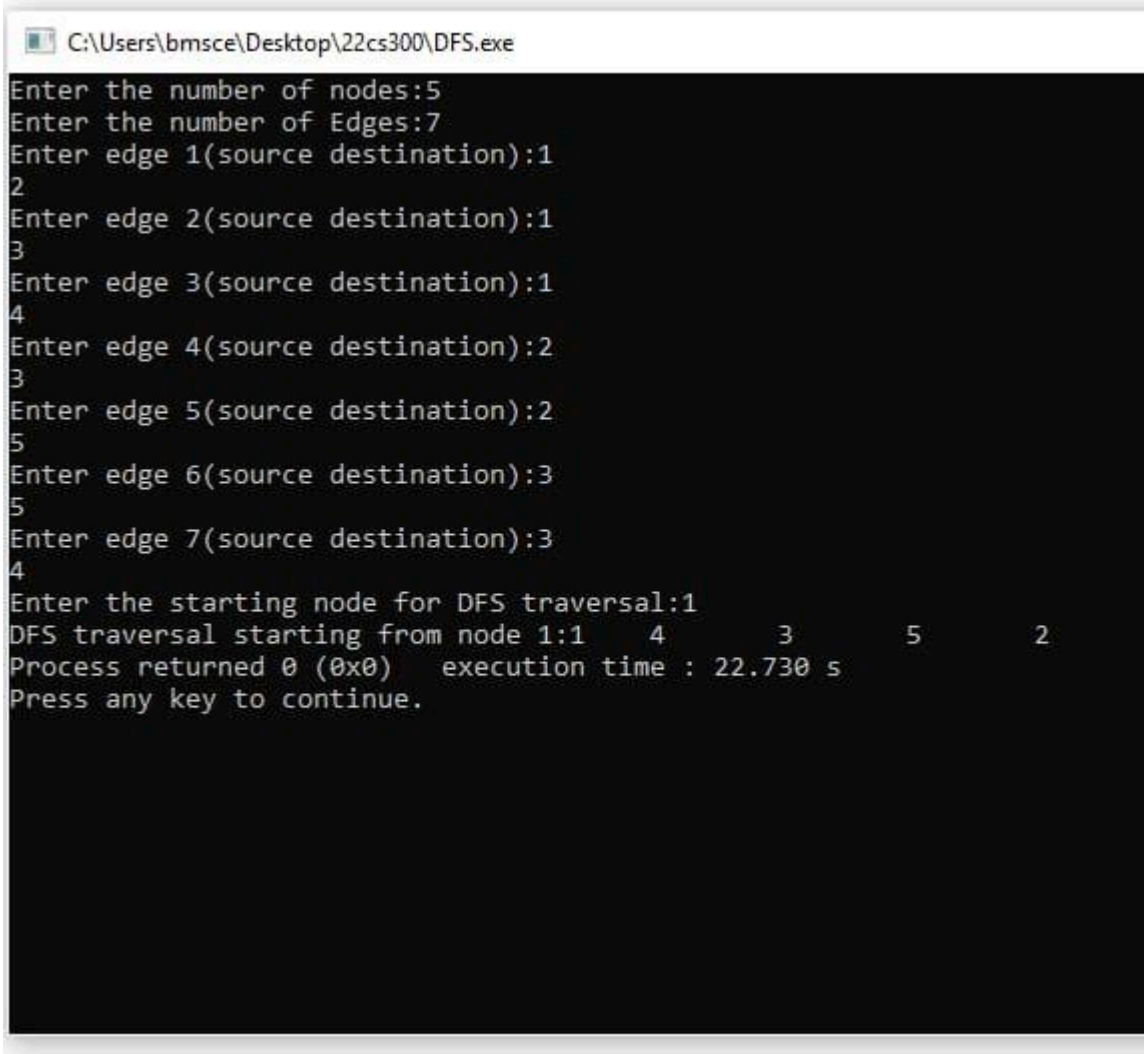
```
struct Node {
    int data;
    struct Node* next;
};

// Define a structure for the graph
struct Graph {
    int numNodes;
    struct Node* adjLists[MAX_NODES];
    int visited[MAX_NODES];
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to create a graph with n nodes
struct Graph* createGraph(int n) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numNodes = n;
    for (int i = 0; i < n; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
```

```
    }
    return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to 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;
}

// Function to perform Depth First Search
void DFS(struct Graph* graph, int startNode) {
    // Mark the current node as visited
    graph->visited[startNode] = 1;
    printf("%d ", startNode);

    // Get all adjacent vertices of the current node
    struct Node* temp = graph->adjLists[startNode];
    while (temp) {
        int adjNode = temp->data;
```

```
        if (!graph->visited[adjNode]) {
            DFS(graph, adjNode);
        }
        temp = temp->next;
    }
}

int main() {
    // Get the number of nodes from the user
    int numNodes;
    printf("Enter the number of nodes: ");
    scanf("%d", &numNodes);

    // Create a graph with the specified number of nodes
    struct Graph* graph = createGraph(numNodes);

    // Get the number of edges from the user
    int numEdges;
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    // Add edges
    for (int i = 0; i < numEdges; i++) {
        int src, dest;
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
```

```
    }

    // Print DFS traversal
    int startNode;
    printf("Enter the starting node for DFS traversal: ");
    scanf("%d", &startNode);
    printf("DFS traversal starting from node %d: ", startNode);
    DFS(graph, startNode);

    return 0;
}
```

OUTPUT

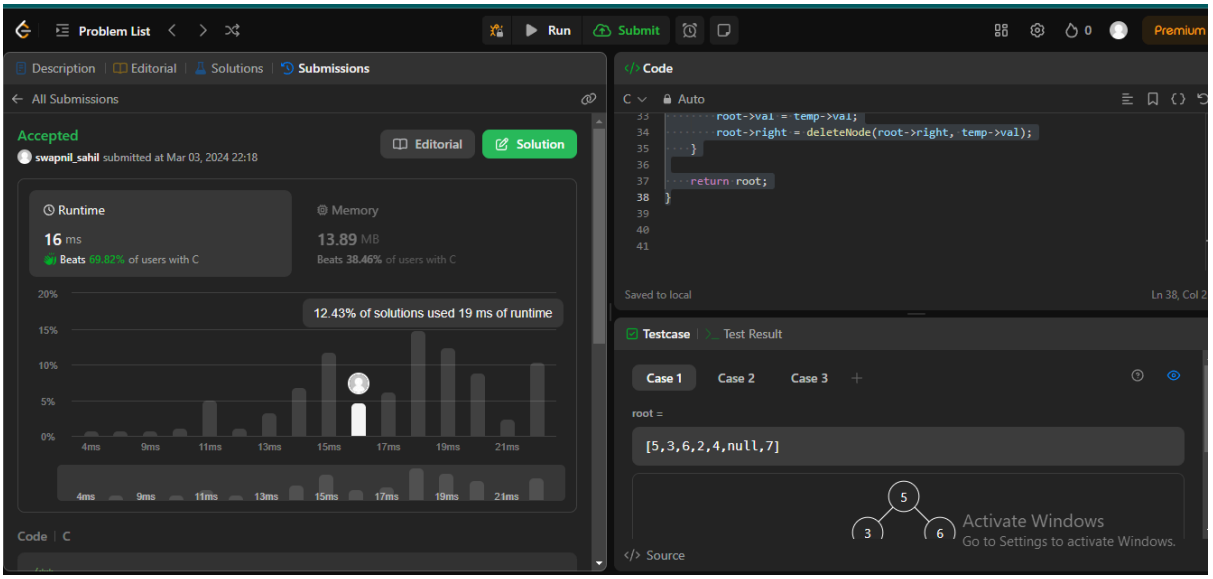


LeetCode Problem:

a)Delete Node In BST

```
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);
```

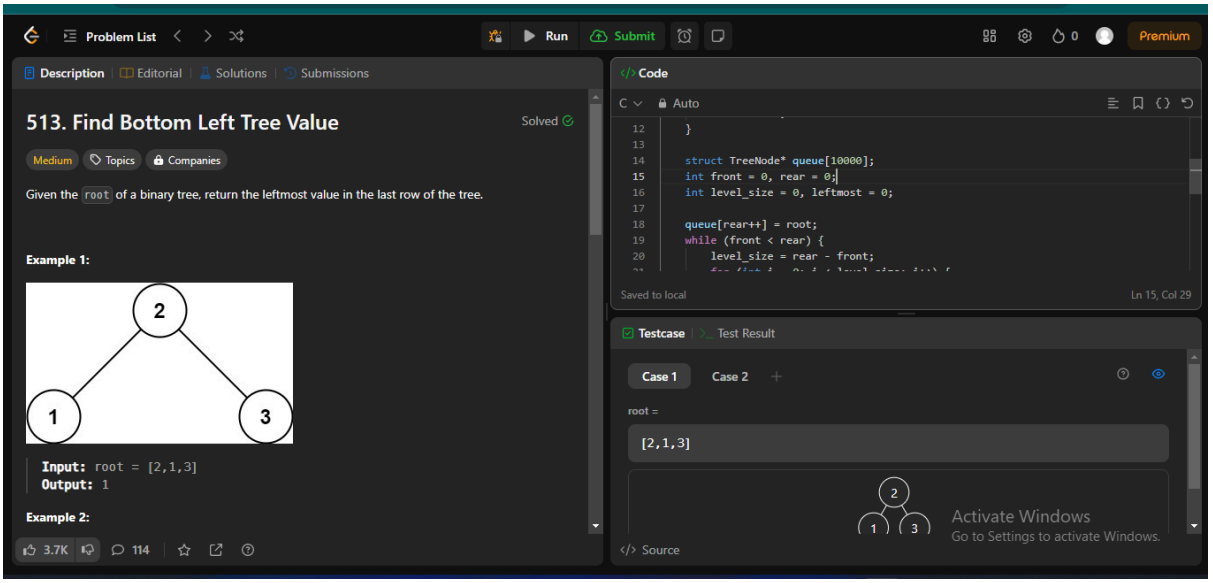
```
    } else {
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }

        struct TreeNode* temp = root->right;
        while (temp->left != NULL) {
            temp = temp->left;
        }
        root->val = temp->val;
        root->right = deleteNode(root->right, temp->val);
    }

    return root;
}
```



b)Find Bottom Left Tree Value

```c
int findBottomLeftValue(struct TreeNode* root) {
    if (root==NULL) {
        return root;
    }

    struct TreeNode* queue[10000];
    int front = 0, rear = 0;
    int level_size = 0, leftmost = 0;

    queue[rear++] = root;
    while (front < rear) {
        level_size = rear - front;
        for (int i = 0; i < level_size; i++) {
            struct TreeNode* node = queue[front++];
            if (i == 0) {
```

```
                leftmost = node->val;

            }

            if (node->left) {

                queue[rear++] = node->left;

            }

            if (node->right) {

                queue[rear++] = node->right;

            }

        }

    }


    return leftmost;

}
```



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>


#define MAX_EMPLOYEES 100

#define HT_SIZE 10


typedef struct {

    int key;

    char name[50];

    // Add other employee details as needed

} Employee;


typedef struct {

    Employee *data;

    int key;

} HashTableEntry;


int hashFunction(int key, int m) {

    return key % m;

}
```

```
void insert(Employee employee, HashTableEntry *hashTable, int m) {
    int index = hashFunction(employee.key, m);
    int i = 1;
    while (hashTable[index].key != -1) {
        index = (index + i) % m;
        i++;
    }
    hashTable[index].key = employee.key;
    hashTable[index].data = &employee;
}

Employee* search(int key, HashTableEntry *hashTable, int m) {
    int index = hashFunction(key, m);
    int i = 1;
    while (hashTable[index].key != -1 && hashTable[index].key != key) {
        index = (index + i) % m;
        i++;
    }
    if (hashTable[index].key == key) {
        return hashTable[index].data;
    } else {
        return NULL;
    }
}

int main() {
    Employee employees[MAX_EMPLOYEES];
```

```
HashTableEntry hashTable[HT_SIZE];
for (int i = 0; i < HT_SIZE; i++) {
    hashTable[i].key = -1;
}

// Assume employees are read from a file and stored in the employees array

int n; // Number of employees
printf("Enter the number of employees: ");
scanf("%d", &n);

for (int i = 0; i < n; i++) {
    printf("Enter employee key and name (separated by a space): ");
    scanf("%d %s", &employees[i].key, employees[i].name);
    insert(employees[i], hashTable, HT_SIZE);
}

int searchKey;
printf("Enter the key to search: ");
scanf("%d", &searchKey);
Employee *result = search(searchKey, hashTable, HT_SIZE);
if (result != NULL) {
    printf("Employee found: %s\n", result->name);
} else {
    printf("Employee not found\n");
}
```

```
    return 0;
}
```

OUTPUT: