VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

DATA STRUCTURE LAB RECORD

Submitted by

SHREEHARI KULKARNI (1BM19CS153)

Under the Guidance of

Prof. POOJA S
Assistant Professor, BMSCE

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

Sep-2020 to Jan-2021

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the LAB RECORD carried out by **SHREEHARI KULKARNI (1BM19CS153)** who is the bonafide students 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 Visveswaraiah Technological University, Belgaum during the year 2020-2021. The lab report has been approved as it satisfies the academic requirements in respect of **DATA STRUCTURE LAB RECORD (19CS3PCDST)** work prescribed for the said degree.

Signature of the Guide
Prof. Prof. POOJA S
Assistant Professor
BMSCE, Bengaluru

Signature of the HOD
Dr. Umadevi V
Associate Prof.& Head, Dept. of CSE
BMSCE, Bengaluru

External Viva

Name of the Examiner — Signature with date

1._____ _____

2. _____ _____

INDEX

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

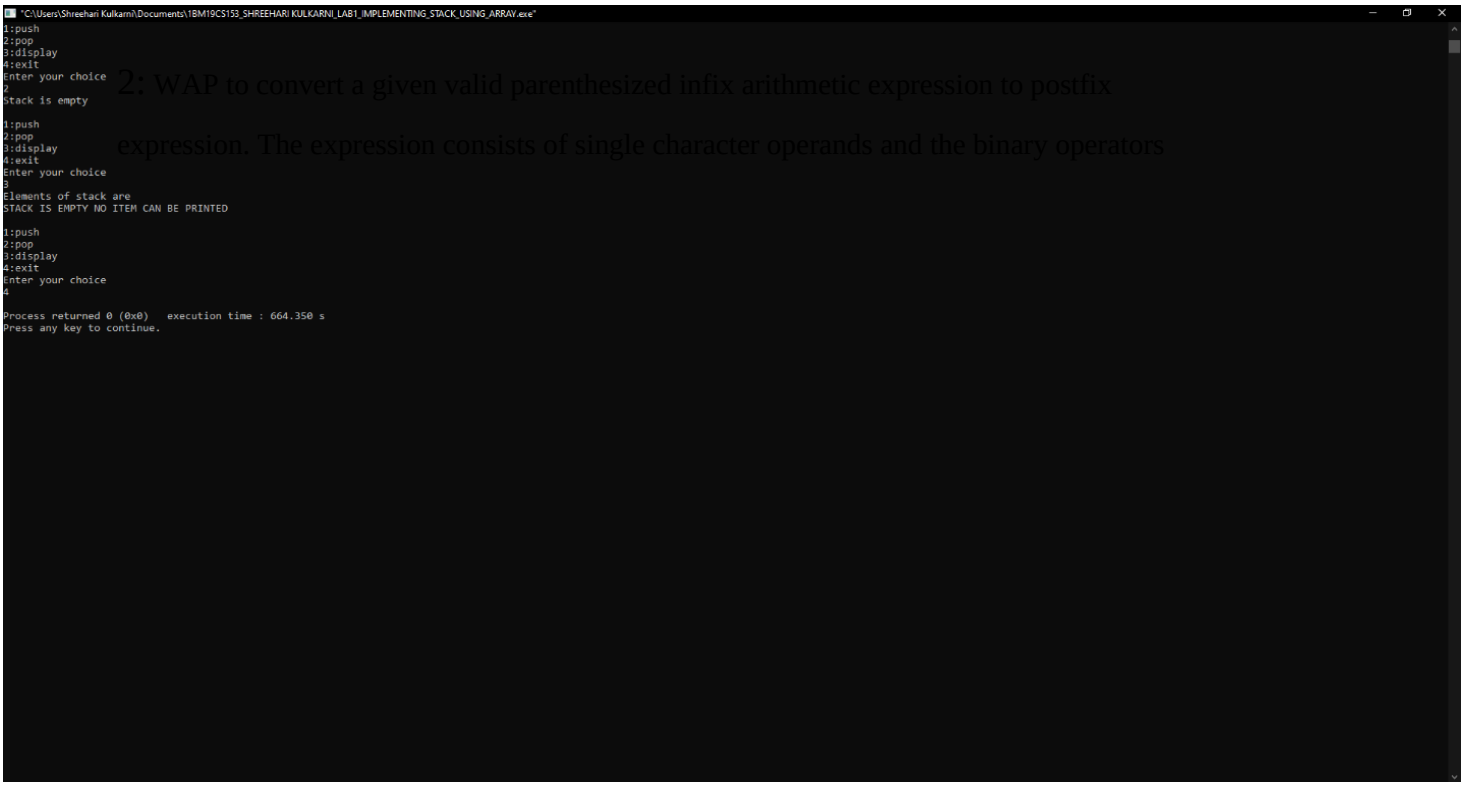
```
#include<stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
int top=-1;
int s[10];
int item;
void push()
{
   if(top==STACK_SIZE-1)
   {

      printf("STACK OVERFLOW ELEMENT CANNOT BE ADDED\n");
      return;
   }
   top+=1;
   s[top]=item;
}

int pop()
```

```
{
   if(top==-1)
   {
      return -1;
   }
   return s[top--];
}
 void display()
 {

    if(top==-1)
    {
       printf("STACK IS EMPTY NO ITEM CAN BE PRINTED\n");
       return;
    }
    else
    {
       for(int i=top;i>=0;i--)
       {
          printf("%d\n",s[i]);
       }
    }

 }
 int main()
 {
```

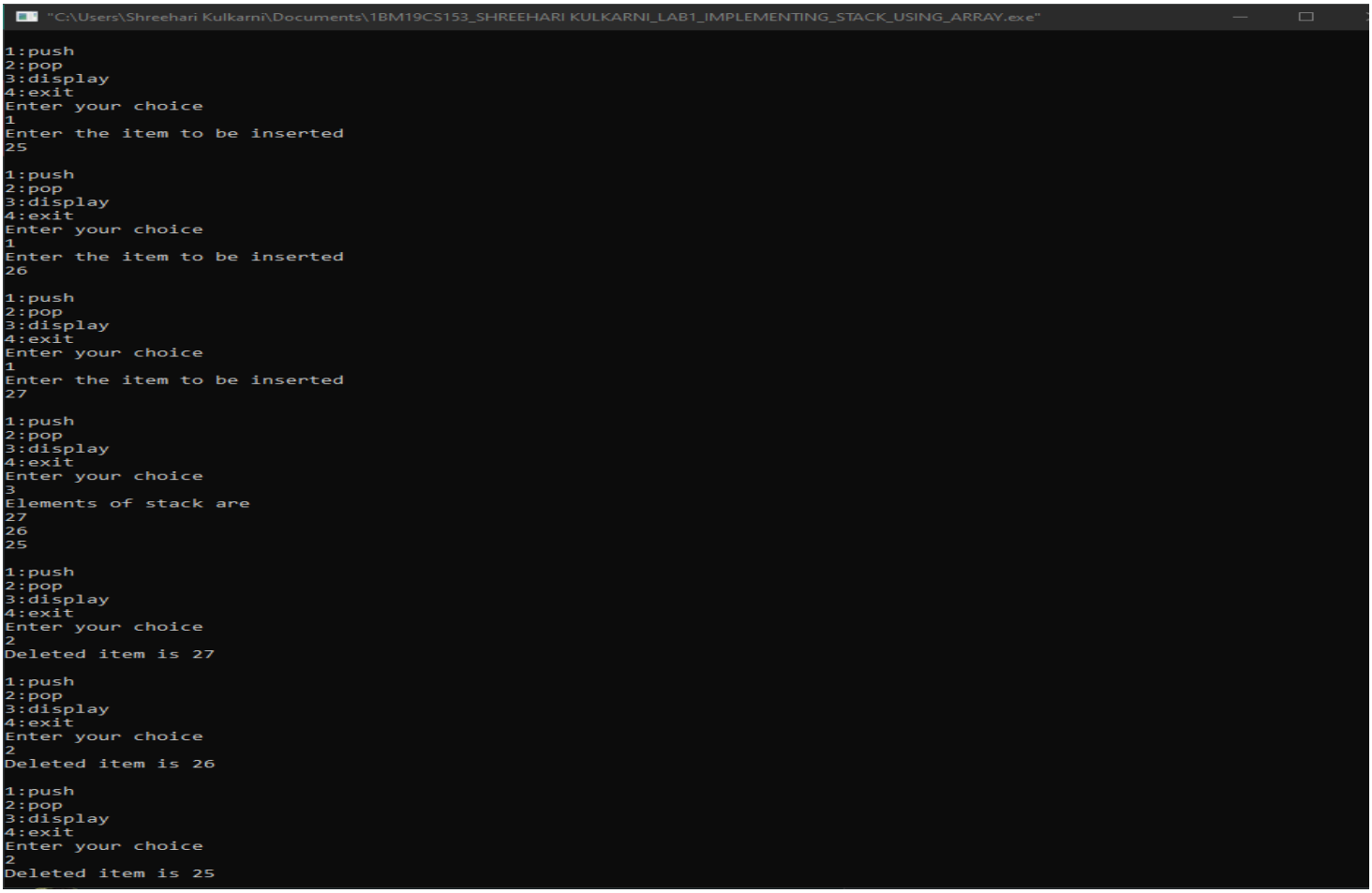
```
int item_deleted;
int choice;
for(;;)
{
   printf("\n1:push\n2:pop\n3:display\n4:exit\n");
   printf("Enter your choice\n");
   scanf("%d",&choice);
   switch(choice)
   {

   case 1:
      printf("Enter the item to be inserted\n");
      scanf("%d",&item);
      push();
      break;

   case 2:
      item_deleted=pop();
      if(item_deleted==-1)
         printf("Stack is empty\n");
      else{
         printf("Deleted item is %d\n",item_deleted);
      }
      break;

   case 3:
```

```
            printf("Elements of stack are \n");
            display();
            break;


        default:
            exit(0);
        }
    }

}
```

1:OUTPUT:



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



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>
#include<stdlib.h>
char stack[100];
int top1=-1;
/*code to check validity of Expression*/
int check(char a[])
{
   for (int i = 0; a[i] != '\0';i++)
        {
                if (a[i] == '(')
                {
                        push(a[i]);
                }
                else if (a[i] == ')')
                {
                        pop();
```

```
            }
        }
        return(find_top());

}
void push(char a)
{
        stack[++top1] = a;
}
void pop()
{
        if (top1 == -1)
        {
                printf("expression is invalid\n");
                exit(0);
        }
        else
        {
                top1--;
        }
}
int find_top()
{
        if (top1 == -1)
                return 1;
        else
```

```
        return 0;
}

/*END OF CHECK FOR INVALID EXPRESSION*/
int F(char symbol)
{
   switch(symbol)
   {

   case '+':
   case '-':
      return 2;



   case '*':
   case '/':
      return 4;



   case '^':
   case '$':
      return 5;

   case '(':
      return 0;
```

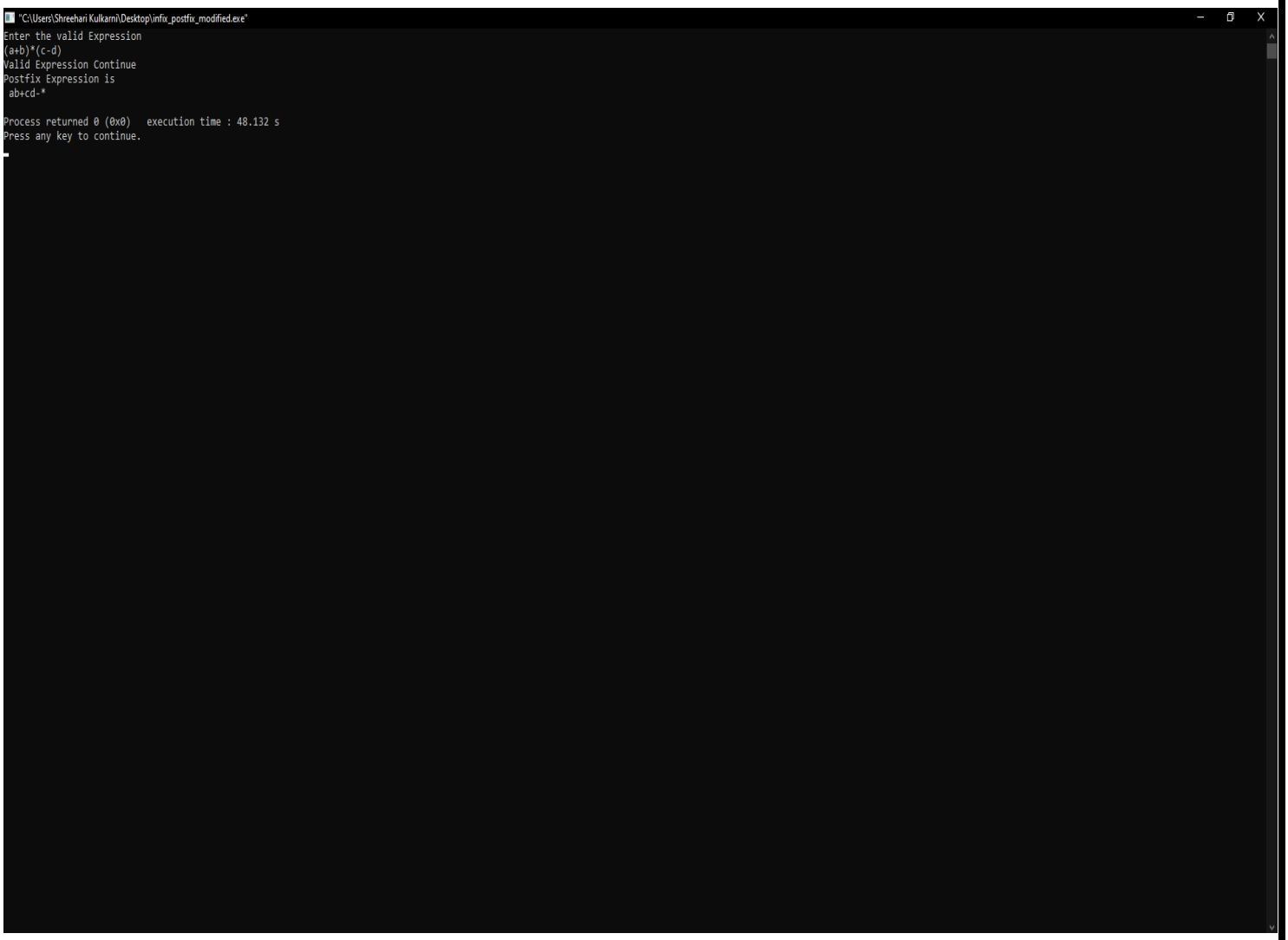
```
    case '#':
        return -1;


    default:
        return 8;

    }
}

int G(char symbol)
{

    switch(symbol)
    {

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


    case '*':
    case '/':
        return 3;
```

```
    case '^':
    case '$':
        return 6;

    case '(':
        return 9;

    case ')':
        return 0;

    default:
        return 7;
    }
}

void  infix_postfix(char infix[],char postfix[])
{

    int top,i,j;
    char s[30],symbol;
    top=-1;
    s[++top]='#';
    j=0;
     if(check(infix))
     {
```

```
    printf("Valid Expression Continue\n");
 }
 else
 {
    printf("Invalid Expression\n");
    exit(0);
 }
for(i=0;i<strlen(infix);i++)
{
   symbol=infix[i];
   while(F(s[top])>G(symbol))
   {
      postfix[j]=s[top--];
      j++;
   }

   if(F(s[top])!=G(symbol))
   {
      s[++top]=symbol;
   }
   else
   {
      top--;
   }
}
```

```
    while(s[top]!='#')
    {
        postfix[j++]=s[top--];
    }
    postfix[j]='\0';
}

int main()
{

    char infix[20];
    char postfix[20];
    printf("Enter the valid Expression\n");
    scanf(" %s",infix);
    infix_postfix(infix,postfix);
    fflush(stdin);
    printf("Postfix Expression is\n");
    printf(" %s\n",postfix);

    return 0;

}
```

2: OUTPUT


"C:\Users\Shreehari Kulkarni\Desktop\infix_postfix_modified.exe"
Enter the valid Expression
(a+b)*(c-d)
Valid Expression Continue
Postfix Expression is
ab+cd-*
Process returned 0 (0x0) execution time : 48.132 s
Press any key to continue.

3: 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>
#include<math.h>
#include<stdlib.h>
#define max 5
int q[max];
int front=-1,rear=-1;
void insert();
int delete();
void display();
int main()
{

   int option,val;
   do
   {
```

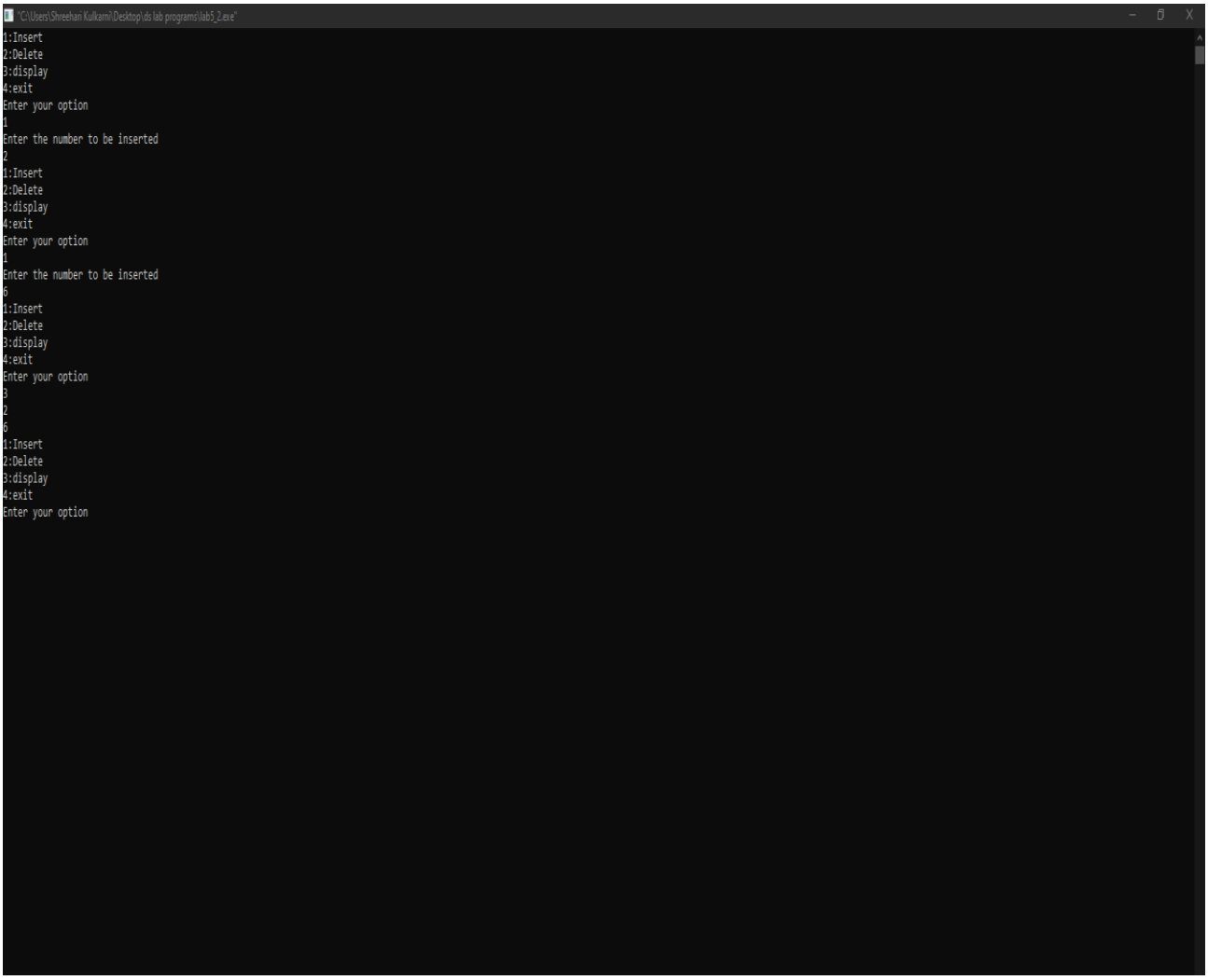
```
printf("**********MAIN MENU**********\n");
printf("1:INSERT\n");
printf("2:DELETE\n");
printf("3:DISPLAY\n");
printf("4:EXIT\n");
printf("Enter the option\n");
scanf("%d",&option);
switch(option)
{

case 1:
   insert();
   break;
case 2:
   val=delete();
   if(val!=-1)
   {
      printf("Item deleted is %d\n",val);
   }
   break;
case 3:
   display();
   break;

case 4:
   exit(0);
```

```
        }
    }while(option!=5);
    return 0;
}

void insert()
{
    int num;
    printf("Enter the number to be inserted\n");
    scanf("%d",&num);
    if(rear==max-1)
    {
        printf("OVERFLOW\n");
    }
    if(front==-1&&rear==-1)
    {
        front=0;
        rear=0;
        q[rear]=num;
    }
    else
    {
        rear++;
        q[rear]=num;
    }
}
```

```
int delete()
{
   int val;
   if(front==-1||front>rear)
   {
      printf("UNDERFLOW\n");
      return -1;
   }
   else
   {
      val=q[front];
      front++;
      if(front>rear)
      {
         front=-1;
         rear=-1;
      }

      return val;
   }
}
void display()
{
   if(front==-1||front>rear)
   {
      printf("Queue Is Empty\n");
```

```
    }
    else
    {
       for(int i=front;i<=rear;i++)
       {
           printf("%d\n",q[i]);
       }
    }

}
```

3: OUTPUT



4: 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<math.h>
#include<stdlib.h>
#define max 5
int q[max];
int front=-1;
int rear=-1;
void insert();
int del();
void display();
int main()
{
   int option,val;
   do
   {
      printf("1:Insert\n");
      printf("2:Delete\n");
```

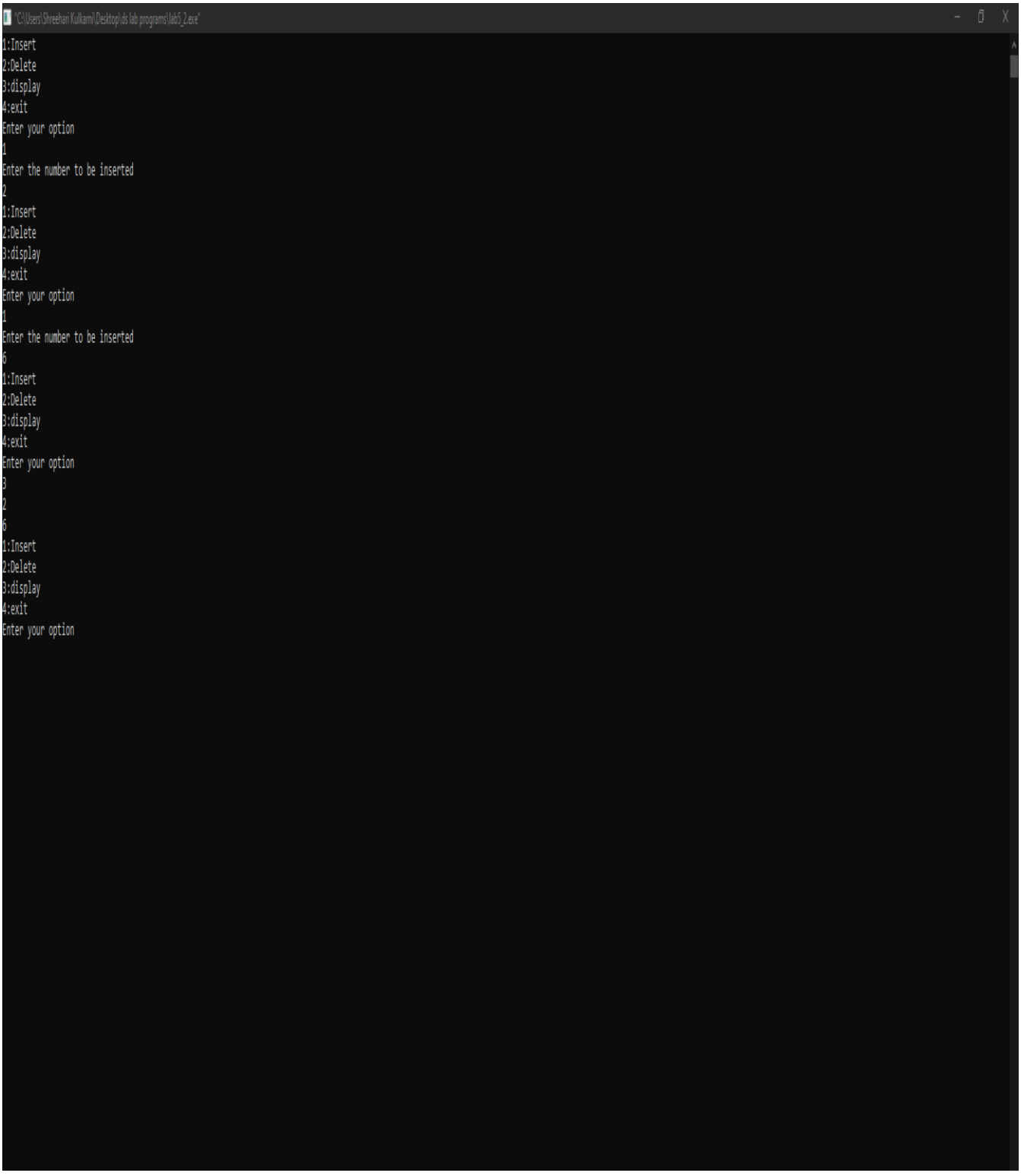
```
        printf("3:display\n");
        printf("4:exit\n");
        printf("Enter your option\n");
        scanf("%d",&option);
        switch(option)
        {
        case 1:
            insert();
            break;
        case 2:
            val=del();
            if(val!=-1)
                printf("Item Deleted is %d\n",val);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        }

    }while(option!=4);
    return 0;
}
void insert()
{
```

```
int num;
printf("Enter the number to be inserted\n");
scanf("%d",&num);
if(front==0&&rear==max-1)
   printf("Overflow\n");
else if(front==-1&&rear==-1)
{
   front=0;
   rear=0;
   q[rear]=num;
}
else if(rear==max-1&&front!=0)
{
   rear=0;
   q[rear]=num;
}
else if(rear==max-1&&front!=0)
{
   rear=0;
   q[rear]=num;
}
else
{
   rear++;
   q[rear]=num;
}
```

```
}

int del()
{
   int val;
   if(front==-1&&rear==-1)
   {
      printf("UNDERFLOW\n");
      return -1;
   }
   val=q[front];
   if(front==rear)
   {
      front=-1;
      rear=-1;
   }
   else
   {
      if(front==max-1)
         front=0;
      else
         front++;
   }
   return val;
}
```

```
void display()
{
   if(front==-1&&rear==-1)
      printf("QUEUQ UNDERFLOW\n");
   else
   {
      if(front<rear)
      {
         for(int i=front;i<=rear;i++)
         {
            printf("%d\n",q[i]);
         }
      }
      else
      {
         for(int i=front;i<max;i++)
            printf("%d\n",q[i]);
         for(int i=0;i<=rear;i++)
            printf("%d\n",q[i]);
      }
   }
}
```

4: OUTPUT


"C:\Users\Shreehari Kulkarni\Desktop\ds lab programs\lab5_2.exe"
1:Insert
2:Delete
3:display
4:exit
Enter your option
1
Enter the number to be inserted
2
1:Insert
2:Delete
3:display
4:exit
Enter your option
1
Enter the number to be inserted
6
1:Insert
2:Delete
3:display
4:exit
Enter your option
3
2
6
1:Insert
2:Delete
3:display
4:exit
Enter your option

5: 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.

6: 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.

7: WAP 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>
#include<string.h>
#include<math.h>
#include<malloc.h>
#include<stdlib.h>
struct node
{
   int item;
   struct node *next;
};
typedef struct node *Node;
Node getNode()
```

```
{
   Node x;
   x=(Node)malloc(sizeof(struct node));
   return x;
}
Node insert_front(Node first,int data)
{
   Node new_node;
   new_node=getNode();
   new_node->item=data;
   new_node->next=NULL;
   if(first==NULL)
   {
      return new_node;
   }
      new_node->next=first;
      first=new_node;
      return first;

}
Node insert_end(Node first,int data)
{
   Node last;
   Node new_node;
   new_node=getNode();
   new_node->item=data;
```

```
    new_node->next=NULL;
    if(first==NULL)
    {
       return new_node;
    }
    last=first;
    while(last->next!=NULL)
    {
      last=last->next;
    }
    last->next=new_node;
    return first;
}
Node insert_pos(Node first,int data,int pos)
{
    Node last;
    Node new_node;
    new_node=getNode();
    new_node->item=data;
    if(first==NULL&&pos==1)
    {
       new_node->next=NULL;
       return new_node;
    }
    else if(pos<1)
    {
```

```
        printf("Invalid Position\n");
        return first;
    }
    else if(pos==1)
    {
        new_node->next=first;
        first=new_node;
        return first;
    }
    else
    {
        last=first;
        for(int i=1;i<pos-1;i++)
        {
            last=last->next;
        }
        new_node->next=last->next;
        last->next=new_node;
        return first;
    }

}
Node delete_front(Node first)
{
    Node temp;
    if(first==NULL)
```

```
    {
        printf("List Is Empty Cannot be deleted\n");
        return first;
    }
    temp=first;
    temp=temp->next;
    free(first);
    return temp;
}
Node delete_end(Node first)
{
    Node prev,cur;
    if(first==NULL)
    {
        printf("List Is Empty And Cannot be deleted\n");
        return first;
    }
    cur=first;
    while(cur->next!=NULL)
    {
        prev=cur;
        cur=cur->next;
    }
    prev->next=NULL;
    free(cur);
    return first;
```

```
}
Node delete_pos(Node first,int pos)
{
   Node prev,cur;
   if(first==NULL)
   {
      printf("List is empty and cannot be deleted\n");
      return first;
   }
   if(pos==1)
   {
      cur=first;
      first=first->next;
      free(cur);
      return first;
   }
   cur=first;
   for(int i=1;i<pos;i++)
   {
      prev=cur;
      cur=cur->next;
   }
   prev->next=cur->next;
   free(cur);
   return first;
}
```

```
void display(Node first)
{

   Node temp;
   if(first==NULL)
   {
      printf("List Is Empty\n");
   }
   for(temp=first;temp!=NULL;temp=temp->next)
   {
      printf("%d\n",temp->item);
   }
}
Node reverse(Node first)
{
   Node cur,prev,temp;
   if(first==NULL)
   {
      printf("List Is Empty\n");
      return first;
   }
   cur=NULL;
   while(first!=NULL)
   {
     temp=first;
     first=first->next;
```

```
      temp->next=cur;
      cur=temp;
    }
    return cur;
}
Node delete_specific(Node first,int key)
{
    Node cur,prev;
    if(first->item==key)
    {
        Node temp=first;
        temp=temp->next;
        free(first);
        return temp;
    }
    cur=first;
    while(cur->item!=key)
    {
        prev=cur;
        cur=cur->next;
    }
    prev->next=cur->next;
    free(cur);
    return first;
}
void sort(Node first)
```

```
{
   int t;
   Node temp;
   if(first==NULL)
   {
      printf("List Is Empty\n");
      return;
   }
   for(Node i=first;i!=NULL;i=i->next)
   {
      for(Node j=i->next;j!=NULL;j=j->next)
      {
         if((i->item)>(j->item))
         {
            t=i->item;
            i->item=j->item;
            j->item=t;
         }
         if((i->item)==(j->item))
         {
            first=delete_specific(first,j->item);
            break;
         }

      }
   }
```

```
}
Node concat(Node first,Node second)
{
   Node cur;
   if(first==NULL)
   {
      first=second;
      return first;
   }
   if(second==NULL)
   {
      return first;
   }

      cur=first;
      while(cur->next!=NULL)
      {
         cur=cur->next;
      }
      cur->next=second;
      printf("Final Concatinated List is \n");
      return first;
}
Node merged_sort(Node a,Node b)
{
   Node result;
```

```
    if(a==NULL)
    {
       return b;
    }
    else if(b==NULL)
    {
       return b;
    }
    if(a->item<=b->item)
    {
       result=a;
       result->next=merged_sort(a->next,b);
    }
    else
    {
       result=b;
       result->next=merged_sort(a,b->next);
    }
    return result;
}
Node order_list(Node first,int data)
{
    Node new_node;
    Node prev,cur;
    new_node=getNode();
    new_node->item=data;
```

```
    if(data<first->item)
    {
       new_node->next=first;
       first=new_node;
       return first;
    }
    cur=first;
    prev=NULL;
    while(cur!=NULL&&cur->item<data)
    {
       prev=cur;
       cur=cur->next;
    }
    prev->next=new_node;
    new_node->next=cur;
    return first;
}

int main()
{
    Node first=NULL;
    Node a= NULL;
    Node b=NULL;
    Node ans=NULL;
    int choice,val,pos,n;
    do
```

```
{
   printf("1:Insert at Front\n");
   printf("2:Insert at End\n");
   printf("3:Insert at Specified Position\n");
   printf("4:Delete Front\n");
   printf("5:Delete End\n");
   printf("6:Delete At Specified Position\n");
   printf("7:Display\n");
   printf("8:Reverse the list\n");
   printf("9:Sort the List\n");
   printf("10:Concat\n");
   printf("11: Ordered List\n");
   printf("12: Merged Sort\n");
   printf("13: Exit\n");*/
   printf("Enter your choice\n");
   scanf("%d",&choice);
   switch(choice)
   {
   case 1:
      printf("Enter the Value To Be Inserted\n");
      scanf("%d",&val);
      first=insert_front(first,val);
      break;
   case 2:
      printf("Enter the Value\n");
      scanf("%d",&val);
```

```
        first=insert_end(first,val);
        break;
    case 3:
        printf("Enter the Value\n");
        scanf("%d",&val);
        printf("Enter the position to be inserted\n");
        scanf("%d",&pos);
        first=insert_pos(first,val,pos);
        break;
    case 4:
        first=delete_front(first);
        break;
    case 5:
        first=delete_end(first);
        break;
    case 6:
        printf("Enter the position from which element has to be deleted\n");
        scanf("%d",&pos);
        first=delete_pos(first,pos);
        break;
    case 7:
        display(first);
        break;
    case 8:
        first=reverse(first);
        break;
```

```
case 9:
    sort(first);
    break;
case 10:
    printf("Enter the Number of nodes In List 1\n");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        printf("Enter the Item To Be Inserted\n");
        scanf("%d",&val);
        a=insert_end(a,val);
    }
    printf("Enter the Number of nodes In List 2\n");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        printf("Enter the Item To Be Inserted\n");
        scanf("%d",&val);
        b=insert_end(b,val);
    }
    a=concat(a,b);
    display(a);
    break;
case 11:
    printf("Enter the Value\n");
    scanf("%d",&val);
```

```
        first=order_list(first,val);
        break;

    case 12:
        printf("Enter the Number of Nodes in  The List \n");
        scanf("%d",&n);
        printf("Enter the Items In Ascending order for list 1\n");
        for(int i=0;i<n;i++)
        {
            printf("Enter the item %d to be inserted at List 1 \n",(i+1));
            scanf("%d",&val);
            a=insert_end(a,val);
        }
        printf("Enter the item in ascending order for list 2\n");
        for(int i=0;i<n;i++)
        {
            printf("Enter the item %d to be inserted at List 1 \n",(i+1));
            scanf("%d",&val);
            b=insert_end(b,val);
        }
        ans=merged_sort(a,b);
        display(ans);
        break;

    }
}while(choice!=13);
```

}

OUTPUT:


"C:\Users\Shreehari Kulkarni\OneDrive\Desktop\LAB SUBJECT\DS LAB\final_linked_list.exe"
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice
1
Enter the Value To Be Inserted
15
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice
1
Enter the Value To Be Inserted
25
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice
2
Enter the Value
35
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice
2
Enter the Value
59
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice
7
25
15
35
59
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position


"C:\Users\Shreehari Kulkarni\OneDrive\Desktop\LAB SUBJECT\DS LAB\final_linked_list.exe"
7:Display
Enter your choice
4
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice
7
15
15
59
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice
5
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice
7
15
35
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
Enter your choice


"C:\Users\Shreehari Kulkarni\OneDrive\Desktop\LAB SUBJECT\DS LAB\final_linked_list.exe"
Enter your choice
7
45
15
10
4
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
8:Reverse the list
9:Sort the List
10:Concat
11: Ordered List
12: Merged Sort
13: Exit
Enter your choice
9
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
8:Reverse the list
9:Sort the List
10:Concat
11: Ordered List
12: Merged Sort
13: Exit
Enter your choice
7
4
15
25
30
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
8:Reverse the list
9:Sort the List
10:Concat
11: Ordered List
12: Merged Sort
13: Exit
Enter your choice
8
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
8:Reverse the list


"C:\Users\Shreeharı Kulkarnı\OneDrive\Desktop\LAB SUBJECT\DS LAB\final_linked_list.exe"
6:Delete At Specified Position
7:Display
8:Reverse the list
9:Sort the List
10:Concat
11: Ordered List
12: Merged Sort
13: Exit
Enter your choice
1
10
25
15
4
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
8:Reverse the list
9:Sort the List
10:Concat
11: Ordered List
12: Merged Sort
13: Exit
Enter your choice
12
Enter the Number of Nodes in The List
2
Enter the Items In Ascending order for list 1
Enter the item 1 to be inserted at List 1
5
Enter the item 2 to be inserted at List 1
10
Enter the item in ascending order for list 2
Enter the item 1 to be inserted at List 1
2
Enter the item 2 to be inserted at List 1
7
2
5
7
1:Insert at Front
2:Insert at End
3:Insert at Specified Position
4:Delete Front
5:Delete End
6:Delete At Specified Position
7:Display
8:Reverse the list
9:Sort the List
10:Concat
11: Ordered List
12: Merged Sort
13: Exit
Enter your choice
7
10
25
15
4

8: WAP to implement Stack & Queues using Linked Representation

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
struct node
{
   int item;
   struct node *next;
};
typedef struct node *Node;
Node getNode()
{
   Node x;
   x=(Node)malloc(sizeof(struct node));
   return x;
}
Node insert_front(Node first,int data)
{
   Node new_node;
   new_node=getNode();
   new_node->item=data;
   new_node->next=NULL;
```

```
    if(first==NULL)
    {
       return new_node;
    }
       new_node->next=first;
       first=new_node;
       return first;

}
Node insert_end(Node first,int data)
{
    Node last;
    Node new_node;
    new_node=getNode();
    new_node->item=data;
    new_node->next=NULL;
    if(first==NULL)
    {
       return new_node;
    }
    last=first;
    while(last->next!=NULL)
    {
      last=last->next;
    }
    last->next=new_node;
```

```
    return first;
}
Node delete_front(Node first)
{
    Node temp;
    if(first==NULL)
    {
        printf("List Is Empty Cannot be deleted\n");
        return first;
    }
    temp=first;
    temp=temp->next;
    free(first);
    return temp;
}
Node delete_end(Node first)
{
    Node prev,cur;
    if(first==NULL)
    {
        printf("List Is Empty And Cannot be deleted\n");
        return first;
    }
    cur=first;
    while(cur->next!=NULL)
    {
```

```
        prev=cur;
        cur=cur->next;
    }
    prev->next=NULL;
    free(cur);
    return first;
}
void display(Node first)
{

    Node temp;
    if(first==NULL)
    {
        printf("List Is Empty\n");
    }
    for(temp=first;temp!=NULL;temp=temp->next)
    {
        printf("%d\n",temp->item);
    }
}
void stack()
{
    int choice,data;
    Node head=NULL;
    printf("STACK IS IMPLEMENTED INSERT REARE AND DELETE
REAR\n");
```

```
    do{
    printf("1:INSERT REAR\n");
    printf("2:DELETE REAR\n");
    printf("3:DISPLAY\n");
    printf("4:EXIT\n");
    printf("Enter Your Choice\n");
    scanf("%d",&choice);
    switch(choice)
    {

    case 1:
       printf("Enter The Data To Be Inserted\n");
       scanf("%d",&data);
       head=insert_end(head,data);
       break;
    case 2:
       head=delete_end(head);
       break;
    case 3:
       display(head);
       break;
    }

    }while(choice!=4);

}
```

```
void q()
{
   int choice,data;
   Node head=NULL;
   printf("QUEUE IS IMPLEMENTED INSERT FRONT AND DELETE
REAR\n");
   do{
   printf("1:INSERT FRONT\n");
   printf("2:DELETE REAR\n");
   printf("3:DISPLAY\n");
   printf("4:EXIT\n");
   printf("Enter Your Choice\n");
   scanf("%d",&choice);
   switch(choice)
   {

   case 1:
      printf("Enter The Data To Be Inserted\n");
      scanf("%d",&data);
      head=insert_front(head,data);
      break;
   case 2:
      head=delete_end(head);
      break;
   case 3:
      display(head);
```

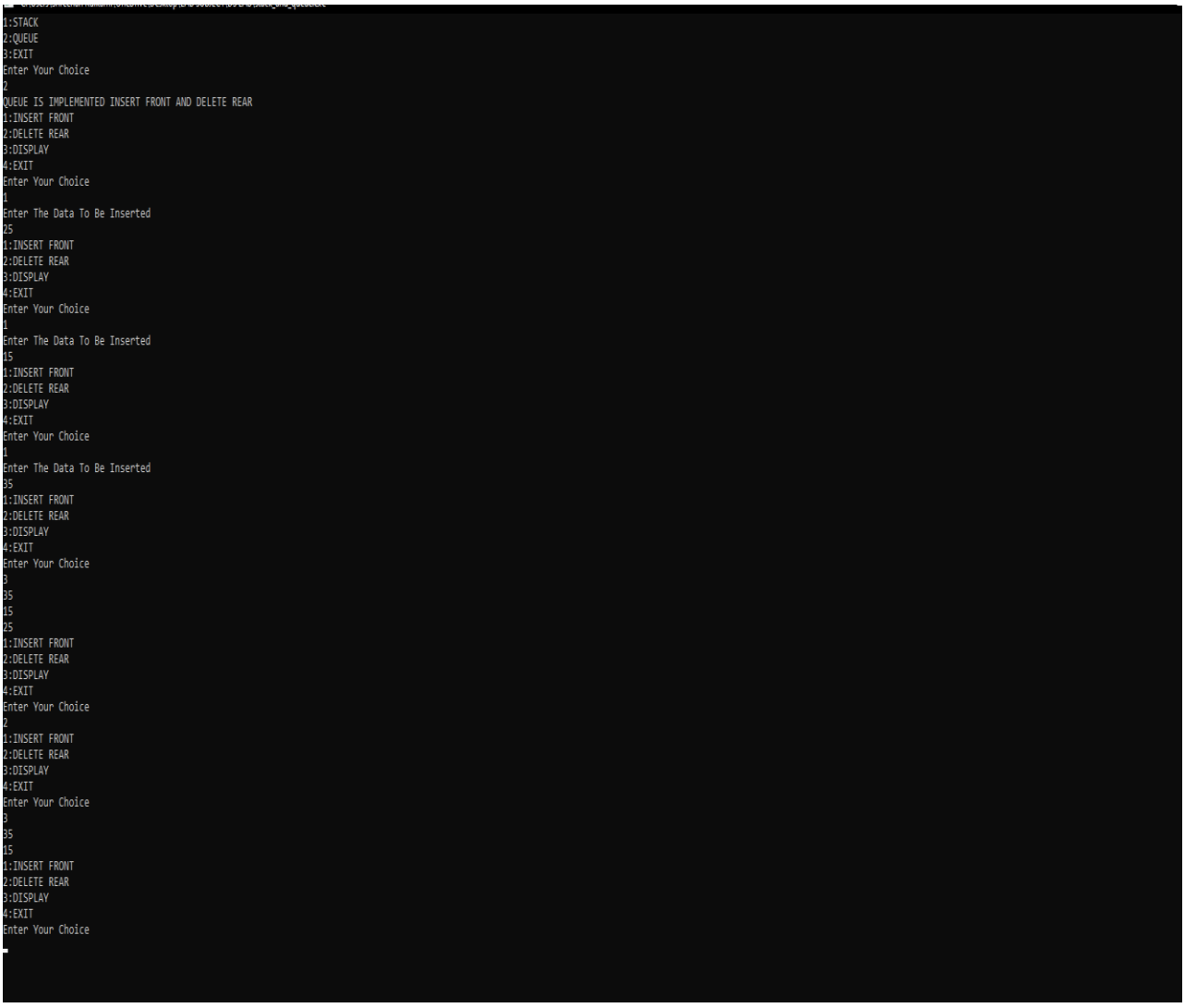
```
        break;
    }

    }while(choice!=4);

}
int main()
{

    int option;
    printf("1:STACK\n");
    printf("2:QUEUE\n");
    printf("3:EXIT\n");
    printf("Enter Your Choice\n");
    scanf("%d",&option);
    switch(option)
    {
    case 1:
        stack();
        break;
    case 2:
        q();
        break;
    case 3:
        exit(0);
        break;
```

```
    }
}
```

OUTPUT:

STACK-OUTPUT:

QUEUE-OUTPUT:



9: WAP 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. c) Display the contents of the list

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
struct node
{
   int info;
   struct node *prev;
   struct node *next;
};
typedef struct node *NODE;
NODE getnode()
{

   NODE x=(NODE)malloc(sizeof(struct node));
   return x;
}
```

```
NODE insert_front(NODE head,int item)
{
   NODE new_node,cur;
   new_node=getnode();
   new_node->info=item;
   cur=head->next;
   head->next=new_node;
   new_node->prev=head;
   new_node->next=cur;
   cur->prev=new_node;
   return head;
}
NODE insert_rear(NODE head,int item)
{
   NODE new_node,cur;
   new_node=getnode();
   new_node->info=item;
   cur=head->prev;
   cur->next=new_node;
   new_node->prev=cur;
   new_node->next=head;
   head->prev=new_node;
   return head;
}
NODE insert_leftpos(NODE head,int key,int item)
{
```

```
int flag=1;
NODE new_node,cur;
new_node=getnode();
new_node->info=item;
cur=head->next;
if(cur->info==key)
{
   new_node->next=cur;
   head->next=new_node;
   new_node->prev=head;
   cur->prev=new_node;
   return head;
   flag=0;
}
else
{
  while(cur->info!=key&&cur!=head)
  {

     cur=cur->next;
      if(cur->info==key)
      {
         flag=0;
      }
  }
  if(flag==0)
```

```
        {
         (cur->prev)->next=new_node;
         new_node->prev=cur->prev;
         new_node->next=cur;
         cur->prev=new_node;
         return head;
        }
        else
        {
            printf("Invalid Key\n");
            return head;
        }

    }
}
NODE insert_rightpos(NODE head,int key,int item)
{
    int flag=1;
    NODE new_node;
    new_node=getnode();
    new_node->info=item;
    NODE cur;
    cur=head->next;
    if(cur->info==key)
    {
        new_node->next=cur->next;
```

```
    (cur->next)->prev=new_node;
    new_node->prev=cur;
    cur->next=new_node;
    flag=0;
    return head;

}
if((cur->info==key)&&(cur->next==head))
{
    cur->next=new_node;
    new_node->prev=cur;
    new_node->next=head;
    head->prev=new_node;
    flag=0;
    return head;
}
while(cur->info!=key)
{
    cur=cur->next;
    if(cur->info==key)
    {
        flag=0;
    }
}
if(flag==0)
{
```

```
        new_node->next=cur->next;
        cur->next=new_node;
        (cur->next)->prev=new_node;
        new_node->prev=cur;
        return head;
    }
    else
    {
        printf("KEY NOT FOUNF\n");
        return head;
    }

}
NODE delete_front(NODE head)
{
    if(head->next==head)
    {
        printf("List Is Empty\n");
        return head;
    }
    NODE temp;
    temp=head->next;
    head->next=temp->next;
    (temp->next)->prev=head;
    free(temp);
    return head;
```

```
}
NODE delete_rear(NODE head)
{
   if(head->next==head)
   {
      printf("List Is Empty\n");
      return head;
   }
   NODE cur;
   cur=head->prev;
   (cur->prev)->next=head;
   head->prev=cur->prev;
   free(cur);
   return head;
}
NODE delete_all(NODE head)
{
   NODE temp;
   for(NODE i=head->next;i!=head;i=i->next)
   {
      for(NODE j=i->next;j!=head;j=j->next)
      {
         if(i->info==j->info)
         {
            temp=j;
            (j->prev)->next=j->next;
```

```
            if(j->next!=head)
            {
               j->next->prev=j->prev;
            }
            temp=NULL;
           // temp=NULL;
         }
      }
   }
}
void display(NODE head)
{
   if(head->next==head)
   {
      printf("List Is Empty\n");
      return;
   }
   NODE cur=head->next;
   while(cur!=head)
   {
      printf("%d\n",cur->info);
      cur=cur->next;
   }
}
void search(NODE head,int key)
{
```

```
NODE cur;
int i=1;
int flag=0;
if((head->next)->info==key)
{
   flag=1;
   printf("Element Found At Position %d\n " + i);
   return;
}
cur=head->next;
while(cur->info!=key&&cur->next!=head)
{
   cur=cur->next;
   i++;
   if(cur->info==key)
   {
      flag=1;
      break;
   }
}
if(flag==1)
{
   printf("Element Found At %d\n",i);
}
else
{
```

```
        printf("Element Not Found\n");
    }

}
int main()
{
    NODE head=getnode();
    head->next=head;
    head->prev=head;
    int option,item,key;
    do{
    printf("1:Insert At Front\n");
    printf("2:Insert At End\n");
    printf("3:Delete Front\n");
    printf("4:Delete Rear\n");
    printf("5:Display\n");
    printf("6:Insert left\n");
    printf("7:Insert Right\n");
    printf("8:Delete Duplicates\n");
    printf("9:Search For Element\n");
    printf("Enter Your Choice\n");
    scanf("%d",&option);
    switch(option)
    {
    case 1:
        printf("Enter the item\n");
```

```
    scanf("%d",&item);
    head=insert_front(head,item);
    break;
case 2:
    printf("Enter the item\n");
    scanf("%d",&item);
    head=insert_rear(head,item);
    break;
case 3:
    head=delete_front(head);
    break;
case 4:
    head=delete_rear(head);
    break;
case 5:
    display(head);
    break;
case 6:
    printf("Enter the key Element\n");
    scanf("%d",&key);
    printf("Enter the Item\n");
    scanf("%d",&item);
    head=insert_leftpos(head,key,item);
    break;
case 7:
    printf("Enter the key Element\n");
```

```
        scanf("%d",&key);
        printf("Enter the item\n");
        scanf("%d",&item);
        head=insert_rightpos(head,key,item);
        break;
    case 8:
        head=delete_all(head);
        break;
    case 9:
        printf("Enter the Element To Be Searched\n");
        scanf("%d",&key);
        search(head,key);
        break;
    }
    }while(option!=10);
}
```

OUTPUT


"C:\Users\Shreeharı Kulkarnı\OneDrive\Desktop\LAB SUBJECT\DS LAB\doublylinkedlist.exe"
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
1
enter the item at front end
45
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
1
enter the item at front end
50
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
1
enter the item at front end
75
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
1
enter the item at front end
100
1:insert front
2:insert rear


"C:\Users\Shreeharı Kulkarnı\OneDrive\Desktop\LAB SUBJECT\DS LAB\doublylinkedlist.exe"
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
1
enter the item at front end
125
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
5
contents of dq
125
100
75
50
25
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
6
enter the key item
100
enter towards left of 100=5
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
5
contents of dq
125


"C:\Users\Shreehari Kulkarni\OneDrive\Desktop\LAB SUBJECT\DS LAB\doublylinkedlist.exe"
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
9
enter the key item
100
enter towards right of 100=10
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
5
contents of dq
125
5
100
10
75
50
25
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice
8
Enter item which you want to search?
100
item found at location 3
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates


"C:\Users\Shreehari Kulkarni\OneDrive\Desktop\LAB SUBJECT\DS LAB\doublylinkedlist.exe"
Enter item which you want to search?
100
item found at location 3
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert Left
7: remove duplicates
8: Search
9:Insert Right
10:Exit
enter the choice

10: 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<math.h>
#include<string.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *left;
   struct node *right;
};
typedef struct node *NODE;

NODE getnode(int data)
{
   NODE x=(NODE)malloc(sizeof(struct node));
   x->data=data;
   x->right=NULL;
   x->left=NULL;
   return x;
```

```
}
NODE insert(NODE root,int info)
{

   if(root==NULL)
   {
      root=getnode(info);
      return root;
   }
   else if(info<=root->data)
   {
      root->left=insert(root->left,info);
   }
   else
   {
      root->right=insert(root->right,info);
   }
   return root;
}
void preorder(NODE root)
{
   if(root==NULL)
      return;
   printf("%d\t",root->data);
   preorder(root->left);
   preorder(root->right);
```

```
}
void inorder(NODE root)
{
   if(root==NULL)
      return;
   inorder(root->left);
   printf("%d\t",root->data);
   inorder(root->right);
}
void postorder(NODE root)
{
   if(root==NULL)
      return;
   postorder(root->left);
   postorder(root->right);
   printf("%d\t",root->data);
}
NODE findmin(NODE root)
{
   if(root==NULL)
   {
      return NULL;
   }
   else if(root->left==NULL)
   {
      return root;
```

```
    }
    return findmin(root->left);
}
NODE findmax(NODE root)
{
    if(root==NULL)
        return NULL;
    else if(root->right==NULL)
        return root;
    return findmax(root->right);
}
NODE delete_node(NODE root,int info)
{
    if(root==NULL)
    {
        return root;
    }
    else if(info<root->data)
    {
        root->left=delete_node(root->left,info);
    }
    else if(info>root->data)
    {
        root->right=delete_node(root->right,info);
    }
    else
```

```
{
   if(root->left==NULL&&root->right==NULL)
   {
     free(root);
     root=NULL;
     return root;
   }
   else if(root->left==NULL)
   {
      NODE temp=root;
      root=root->left;
      free(temp);
      return root;
   }
   else if(root->right==NULL)
   {
      NODE temp=root;
      root=root->right;
      free(temp);
      return root;
  }
  else
  {
      NODE temp=findmin(root->right);
      root->data=temp->data;
      root->right=delete_node(root->right,temp->data);
```

```
            return root;
        }
    }
}
void display(NODE root,int i)
{
    if(root==NULL)
        return;
    display(root->right,i+1);
    for(int j=1;j<=i;j++)
        printf(" ");
    printf("%d\n",root->data);
    display(root->left,i+1);
}
int getleafcount(NODE root)
{
    NODE current=root;
    if(current==NULL)
        return 0;
    else if(current->right==NULL&&current->left==NULL)
        return 1;
    else
        return (getleafcount(current->left)+getleafcount(current->right));
}
int height(NODE root)
{
```

```
    if(root==NULL)
        return -1;
    int left_height=height(root->left);
    int right_height=height(root->right);
    if(left_height>right_height)
        return left_height+1;
    else
        return right_height+1;
}
NODE search(NODE root,int key)
{
    if(root==NULL)
        return NULL;
    if(root->data==key)
        return root;
    else if(key<root->data)
        search(root->left,key);
    else if(key>root->data)
        search(root->right,key);
    else
        printf("Search UnSuccessfull\n");
}
NODE inorder_successor(NODE root,int data)
{
    if(root==NULL)
    {
```

```
    printf("Tree Empty\n");
    return root;
}
NODE current=search(root,data);
//NODE current=root;
if(current==NULL)
   return NULL;
if(current->right!=NULL)
{
    NODE temp=findmin(current->right);
}
else
{
    NODE successor=NULL;
    NODE ancestor=root;
    while(current!=ancestor)
    {
        if(current->data<ancestor->data)
        {
            successor=ancestor;
            ancestor=ancestor->left;
        }
        else
        {
            ancestor=ancestor->right;
        }
```

```
        }
        return successor;
    }
}
int main()
{
    NODE root=NULL;
    NODE value=NULL;
    int data,option,ans;
    do{
    printf("1:Insert\n");
    printf("2:Delete\n");
    printf("3:Preorder\n");
    printf("4:PostOrder\n");
    printf("5:Inorder\n");
    printf("6:Display\n");
    printf("7:Height Of Tree\n");
    printf("8:Find Maximum\n");
    printf("9:Search\n");
    printf("10:In Order Successor\n");
    printf("11:Exit\n");
    printf("Enter Your Choice\n");
    scanf("%d",&option);
    switch(option)
    {
    case 1:
```

```
    printf("Enter The Data To Be Inserted\n");
    scanf("%d",&data);
    root=insert(root,data);
    break;
case 2:
    printf("Enter the Data To Be Deleted\n");
    scanf("%d",&data);
    root=delete_node(root,data);
    break;
case 3:
    preorder(root);
    printf("\n");
    break;
case 4:
    postorder(root);
    printf("\n");
    break;
case 5:
    inorder(root);
    printf("\n");
    break;
case 6:
    display(root,1);
    break;
case 7:
    ans=height(root);
```

```
        printf("Height Of Tree Is %d\n",ans);
        break;
    case 8:
        value=findmax(root);
        printf("Maximum Value is%d\n",value->data);
        break;
    case 9:
        printf("Enter the Key Value To Be Searched\n");
        scanf("%d",&data);
        value=search(root,data);
        if(root!=NULL)
            printf("Search SuccessFull\n");
        else
            printf("Search Unsuccesfull\n");
        break;
    case 10:
        printf("Enter the key value whose successor you want to find\n");
        scanf("%d",&data);
        root=inorder_successor(root,data);
        printf("Successor is %d\n",root->data);

    }
    }while(option!=11);
}
```

OUTPUT:


"C:\Users\Shreehari Kulkarni\OneDrive\Desktop\LAB SUBJECT\DS LAB\binarysearchtree.exe"
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
1
Enter The Data To Be Inserted
15
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
1
Enter The Data To Be Inserted
10
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
1
Enter The Data To Be Inserted
25
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
1
Enter The Data To Be Inserted
35
1:Insert
2:Delete
3:Preorder


"C:\Users\Shreehari Kulkarni\OneDrive\Desktop\LAB SUBJECT\DS LAB\binarysearchtree.exe"
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
6
35
25
15
10
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
3
15 10 25 35
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
4
10 35 25 15
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
5
10 15 25 35
1:Insert
2:Delete
3:Preorder
4:PostOrder