VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

PANNAGA R BHAT
(1BM22CS189)

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 **PANNAGA R BHAT (1BM22CS189)**, who is a 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. Sneha S Bagalkot	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course outcomes:

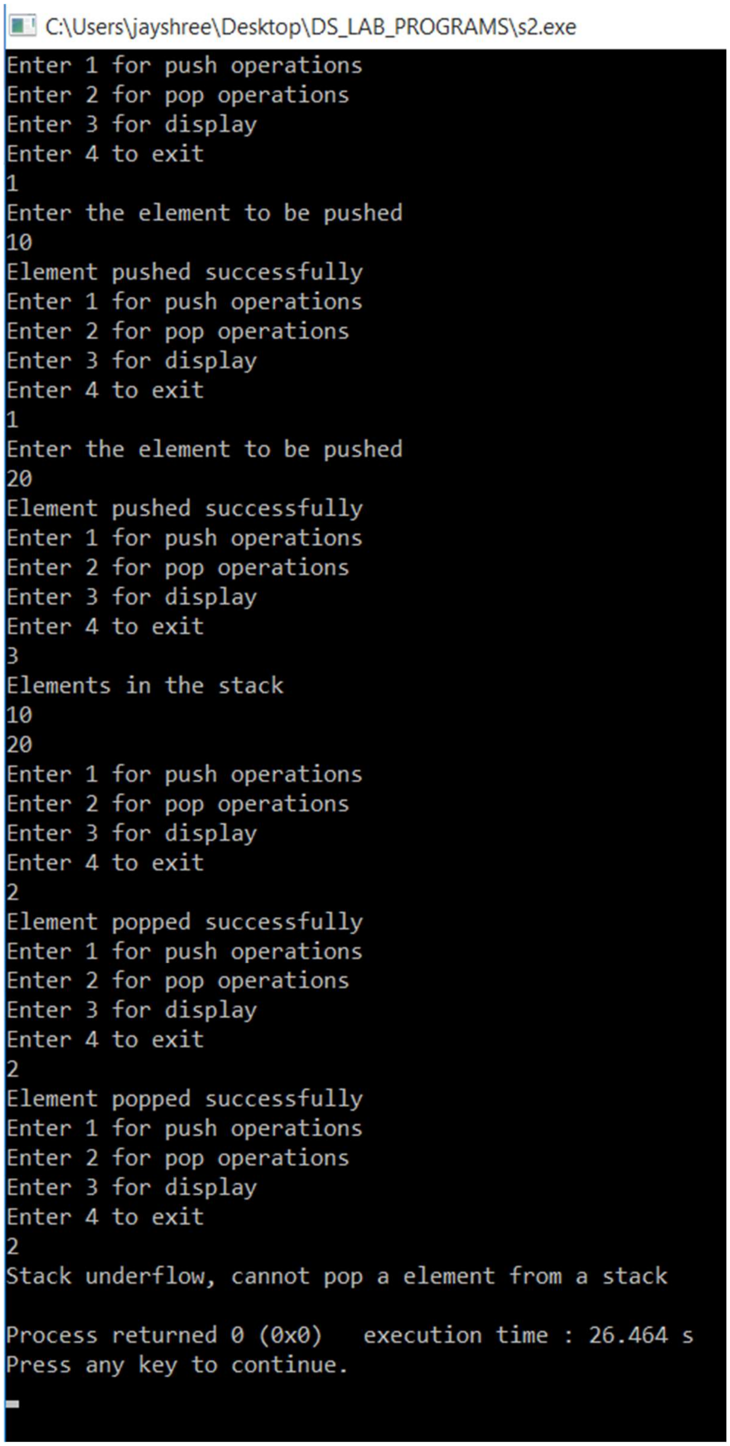
CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations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

void push(int i, int a[]);
void pop();
void display(int a[]);

int top = -1;
int a[size],i;

void main(){
int choice;

while(1){
        printf("Enter 1 for push operations\n");
        printf("Enter 2 for pop operations\n");
        printf("Enter 3 for display\n");
        printf("Enter 4 to exit\n");

        scanf("%d",&choice);
        switch (choice){
        case 1:
           printf("Enter the element to be pushed\n");
           scanf("%d",&i);
           push(i,a);
           break;

        case 2:
           pop();
           break;
        case 3:
           display(a);
           break;
        case 4:
           printf("Programs ends successfully\n");
           exit(0);
        }
    }
}

void push(int i,int a[]){
    if(top == size-1){
        printf("Stack overflow, cannot insert a element into a stack\n");
        exit(0);
    }
    else{
    top = top+1;
    a[top] = i;
    printf("Element pushed successfully\n");
    return;
    }
}

void pop(){
    if(top == -1){
    printf("Stack underflow, cannot pop a element from a stack\n");
    exit(0);
    }
    else{
    top = top-1;
    printf("Element popped successfully\n");
    return;
    }
}

void display(int a[]){

    if(top == -1){
    printf("Stack Underflow");
    exit(0);
    }
    else{
        printf("Elements in the stack\n");
        for(int j=0;j<=top;j++)
        {
           printf("%d\n",a[j]);
        }
        return;
    }
}
```

Output:



C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\s2.exe

```
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
1
Enter the element to be pushed
10
Element pushed successfully
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
1
Enter the element to be pushed
20
Element pushed successfully
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
3
Elements in the stack
10
20
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
2
Element popped successfully
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
2
Element popped successfully
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
2
Stack underflow, cannot pop a element from a stack

Process returned 0 (0x0)   execution time : 26.464 s
Press any key to continue.
```

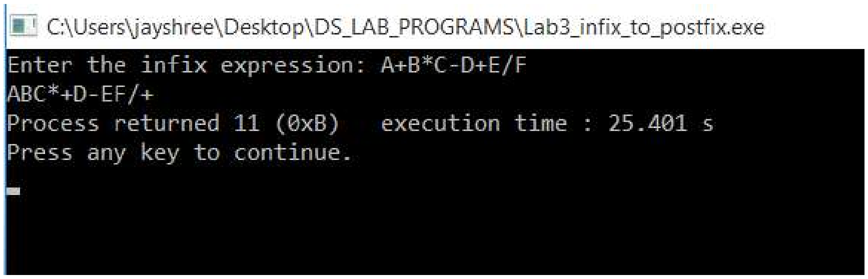
Lab Program 2:

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

```
#include<stdio.h>

#include<ctype.h>

#define max 15

void push(char a);

char pop();

int precedence(char b);


char stack[max];

int top = -1;

void main()

{

    char infix[max], p, postfix[max];

    printf("Enter the infix expression: ");

    scanf("%s",infix);

    int j = 0;

    push('(');

    for(int i=0;i<strlen(infix);i++)

    {

        if(isalnum(infix[i])){

            postfix[j] = infix[i];

            j+=1;

        }

        else if(infix[i]== '+' || infix[i]=='-' || infix[i]=='*' || infix[i]=='/' || infix[i]=='^'){

            if(precedence(infix[i]) > precedence(stack[top]))

                push(infix[i]);
            else if(precedence(infix[i]) <= precedence(stack[top])){
                while(1){
                    p = pop();
                    if (p =='('){
                        push(p);
                        break;
                    }

                    postfix[j] = p;
                    j+=1;
                }
                push(infix[i]);
            }
        }
    }
while(top!=-1){
    char y = pop();
    if (y == '(')
        break;
    postfix[j] = y;
    j+=1;
}
    postfix[j] = '\0';
    printf("%s",postfix);
}
```

```
void push(char a){
   if(top == max-1){
      printf("Stack overflow");
      exit(0);
   }
   else{
      stack[++top] = a;
   }
}

char pop(){
   if(top == -1){
      printf("Stack Underflow");
      exit(0);
   }
   else
      return stack[top--];
}

int precedence(char b){
   if(b == '^')
      return 3;
   else if(b == '/' || b == '*')
      return 2;
   else if(b == '+' || b == '-')
      return 1;
   else
      return 0;
}
```

Output:


C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Lab3_infix_to_postfix.exe

```
Enter the infix expression: A+B*C-D+E/F
ABC*+D-EF/+
Process returned 11 (0xB)   execution time : 25.401 s
Press any key to continue.
```

Lab Program 3:

3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions.

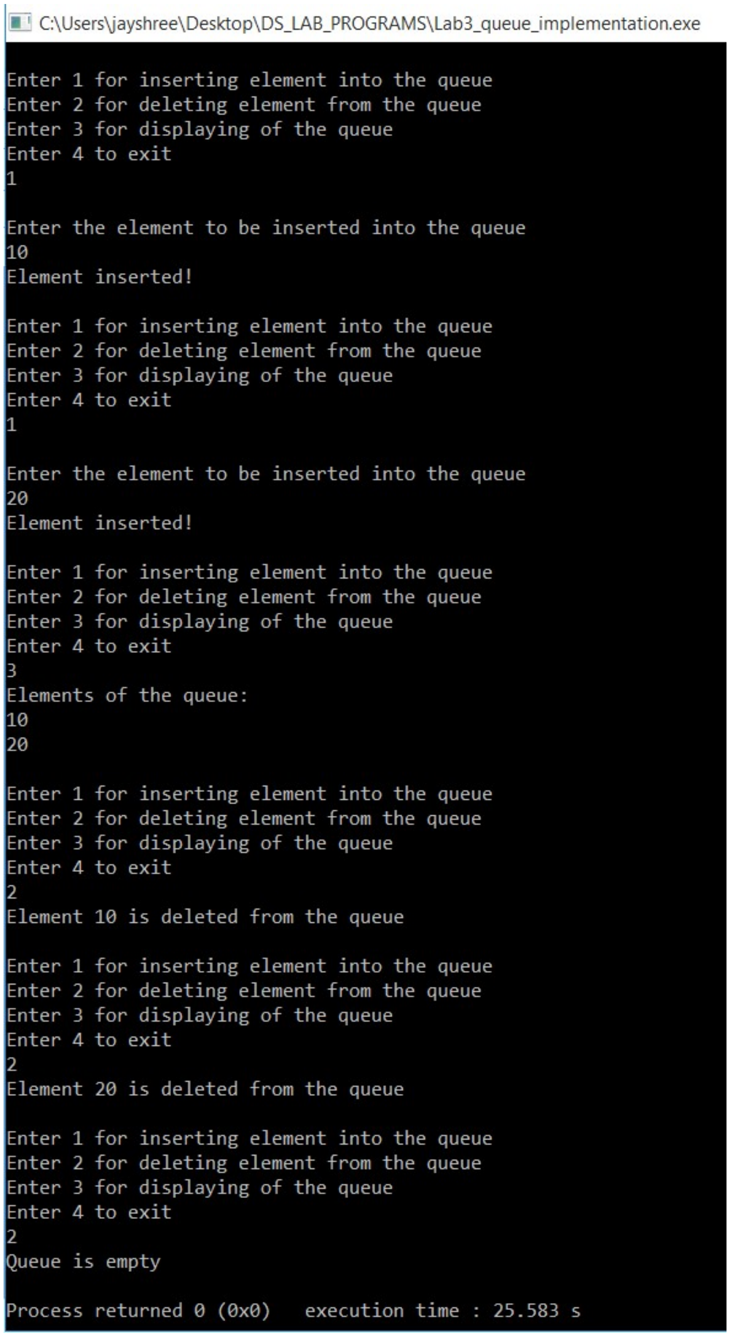
```
#include<stdio.h>

#include<stdlib.h>

#define max 6

void enqueue(int e);

int dequeue();

void display();

int q[max], front = -1, rear = -1, j = 0;


void main(){

   int choice, ele;

   while(1){

      printf("\nEnter 1 for inserting element into the queue\n");

      printf("Enter 2 for deleting element from the queue\n");
```

```
        printf("Enter 3 for displaying of the queue\n");
        printf("Enter 4 to exit\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                printf("\nEnter the element to be inserted into the queue\n");
                scanf("%d",&ele);
                enqueue(ele);
                break;
            case 2:
                printf("Element %d is deleted from the queue\n",dequeue());
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Program ends successfully!!");
                exit(0);
                break;
            default:
                printf("Enter a valid choice.");
                continue;
        }
        continue;
    }
}

void enqueue(int e)
```

```
{
   if(rear == max-1){
       printf("Queue is full\n");
       exit(0);
   }



else
   {   if(rear == -1)
       {
       front=rear=0;
       q[rear] = e;
       }
       else
       {
           rear+=1;
           q[rear] = e;
       }
       printf("Element inserted!\n");
   return;
}
}





int dequeue()
{
   if(front > rear || front == -1){
       printf("Queue is empty\n");
```

```
        exit(0);
    }
    else{
        return q[front++];
    }
}

void display()
{
    if(front == -1 || front > rear){
        printf("Queue is empty\n");
        exit(0);
    }
    else{
        printf("Elements of the queue:\n");
        for(int i=front; i<=rear; i++)
            printf("%d\n",q[i]);
    }
}
```

Output:



C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Lab3_queue_implementation.exe

```

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
1

Enter the element to be inserted into the queue
10
Element inserted!

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
1

Enter the element to be inserted into the queue
20
Element inserted!

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
3
Elements of the queue:
10
20

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
2
Element 10 is deleted from the queue

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
2
Element 20 is deleted from the queue

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
2
Queue is empty

Process returned 0 (0x0)   execution time : 25.583 s
```

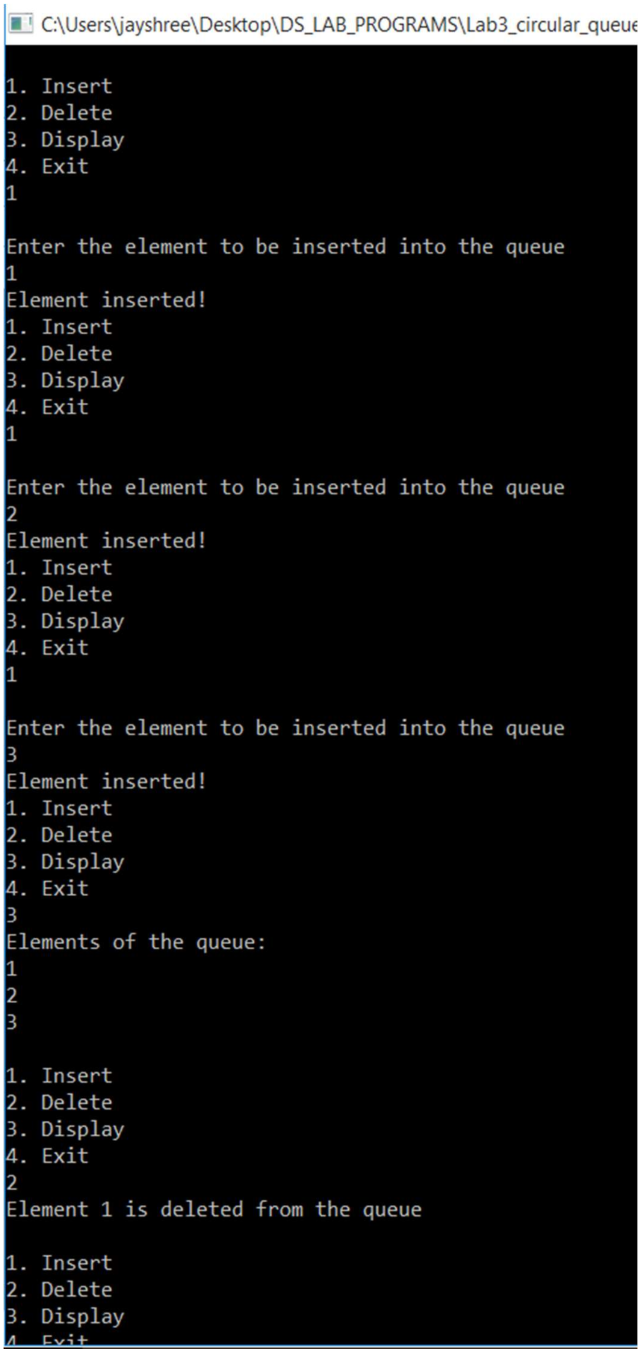
3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include<stdio.h>
#include<stdlib.h>
#define max 3
void enqueue(int e);
int dequeue();
void display();
int q[max], front = -1, rear = -1, j = 0;

void main(){
  int choice, ele;
  while(1){
    printf("\n1. Insert\n");
    printf("2. Delete\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("\nEnter the element to be inserted into the queue\n");
        scanf("%d",&ele);
        enqueue(ele);
        break;
      case 2:
        printf("Element %d is deleted from the queue\n",dequeue());
        break;
```

```
            case 3:
                display();
                break;
            case 4:
                printf("Program ends successfully!!");
                exit(0);
            default:
                printf("Enter a valid choice.");
                continue;
        }
    }
}
void enqueue(int e){
    if((rear-front) == max-1 || (front==(rear+1))){
        printf("Queue is full.");
        exit(0);
    }
   else{
        if(front == -1)
            rear = front = 0;
        else
            rear = (rear+1) % (max);
 }
    q[rear] = e;
    printf("Element inserted!");
}
int dequeue() {
    int item;
    if (front == -1) {
        printf("Queue is empty\n");
        exit(0);
    }
    item = q[front];
if (front == rear) {
        // Only one element in the queue
        front = rear = -1;
    } else {
        front = (front + 1) % max;
    }
    return item;
}
void display() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    if(rear>=front)
    {
        printf("Elements of the queue:\n");
        for(int i=front; i<=rear; i++)
            printf("%d\n", q[i]);
    }
    else{
        printf("Elements of the queue:\n");
       for(int i=front;i<=max-1;i++)
            printf("%d\n", q[i]);


        for(int i=0;i<=rear;i++)
```

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

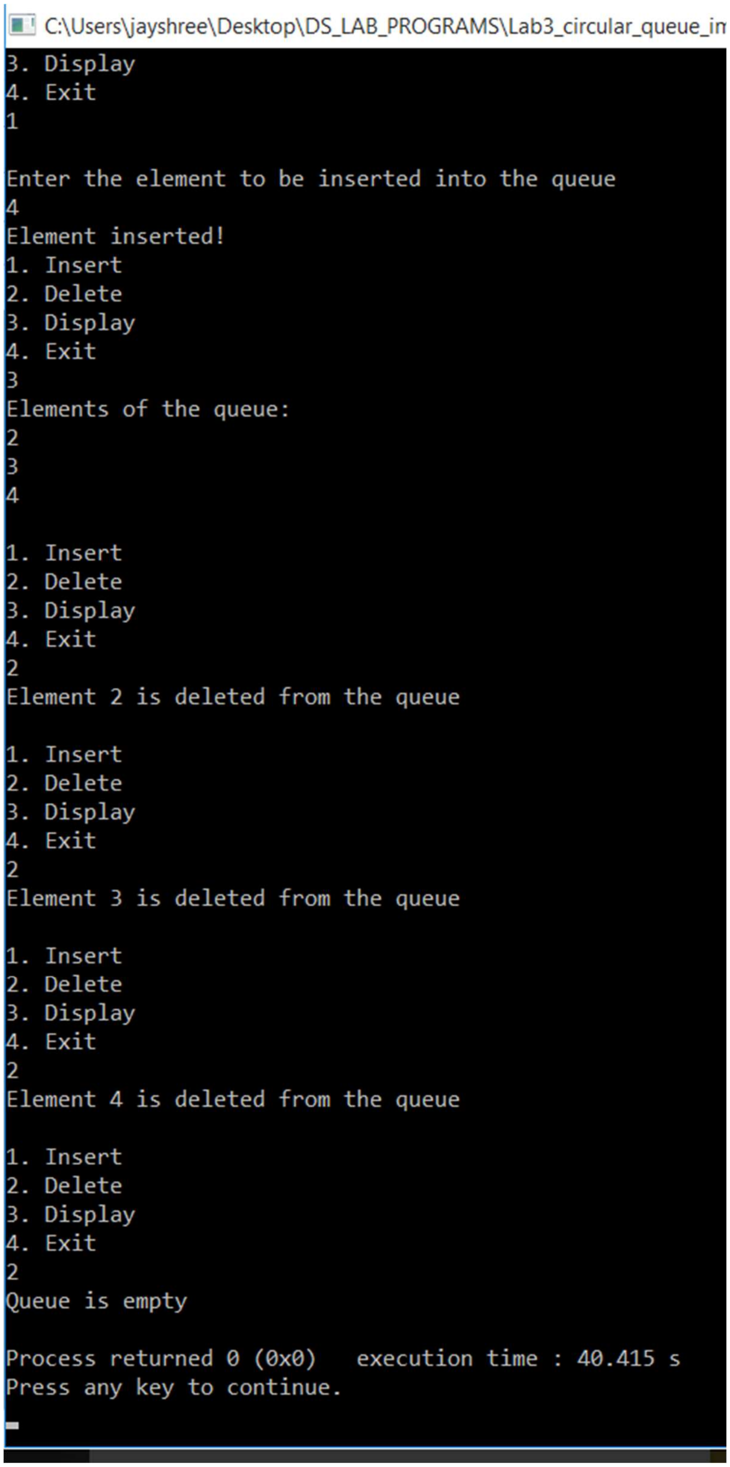
Output: 1)



C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Lab3_circular_queue

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

Enter the element to be inserted into the queue
1
Element inserted!
1. Insert
2. Delete
3. Display
4. Exit
1

Enter the element to be inserted into the queue
2
Element inserted!
1. Insert
2. Delete
3. Display
4. Exit
1

Enter the element to be inserted into the queue
3
Element inserted!
1. Insert
2. Delete
3. Display
4. Exit
3
Elements of the queue:
1
2
3


1. Insert
2. Delete
3. Display
4. Exit
2
Element 1 is deleted from the queue

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

2)



C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Lab3_circular_queue_im

```
3. Display
4. Exit
1

Enter the element to be inserted into the queue
4
Element inserted!
1. Insert
2. Delete
3. Display
4. Exit
3
Elements of the queue:
2
3
4

1. Insert
2. Delete
3. Display
4. Exit
2
Element 2 is deleted from the queue

1. Insert
2. Delete
3. Display
4. Exit
2
Element 3 is deleted from the queue

1. Insert
2. Delete
3. Display
4. Exit
2
Element 4 is deleted from the queue

1. Insert
2. Delete
3. Display
4. Exit
2
Queue is empty

Process returned 0 (0x0)   execution time : 40.415 s
Press any key to continue.
```

Lab Program 4:

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. Display the contents of the linked list.

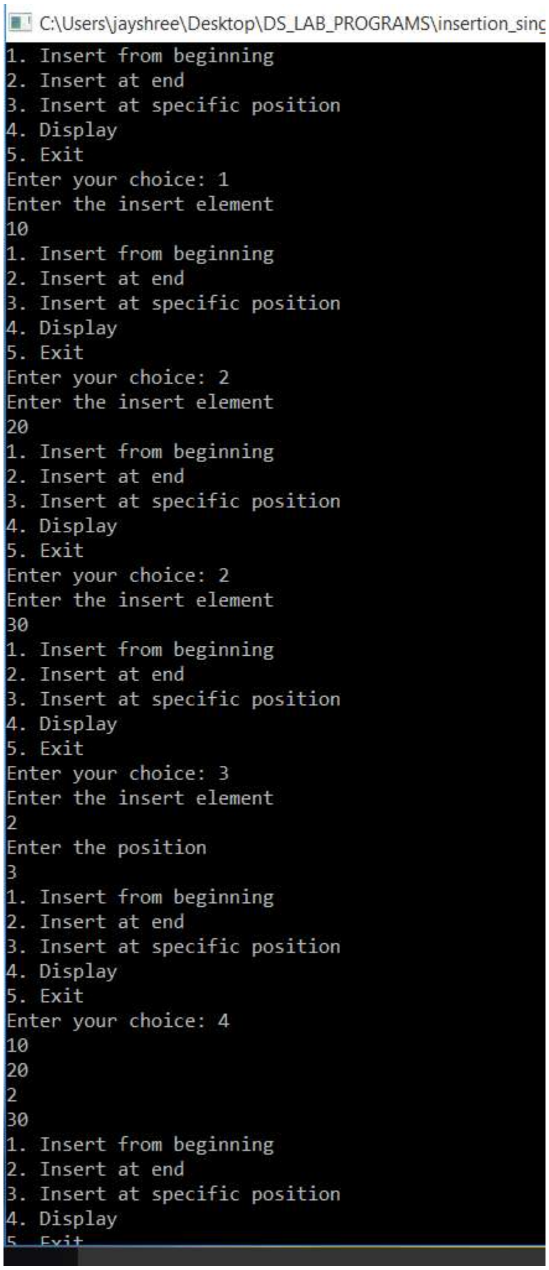
```
#include<stdio.h>

void push(int);

void append(int);

void insert_at_pos(int);

struct Node

{

   int data;

   struct Node *next;

};

struct Node *head = NULL;

int choice,pos,n;


void main()

{

   while(1){

   printf("1. Insert from beginning\n");

   printf("2. Insert at end\n");

   printf("3. Insert at specific position\n");

   printf("4. Display\n");

   printf("5. Exit\n");


   printf("Enter your choice: ");

   scanf("%d",&choice);


   switch(choice)
```

```
    {
    case 1:
        printf("Enter the insert element\n");
        scanf("%d",&n);
        push(n);
        break;
    case 2:
        printf("Enter the insert element\n");
        scanf("%d",&n);
        append(n);
        break;
    case 3:
        printf("Enter the insert element\n");
        scanf("%d",&n);
        insert_at_pos(n);
        break;
    case 4:
        display();
        break;
    case 5:
        exit(0);
    default:
        printf("Enter correct choice");
        break;
        }
        continue;
    }
}
void push(int n)
```

```
{
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   new_node->data = n;
   new_node->next = head;
   head = new_node;
}


void append(int n)
{
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   struct Node* last  = head;  // if head is linked to one or more nodes already, then point last
pointer to head.
   new_node->data = n;
   new_node->next = NULL;
   // if the list is empty.
   if(head == NULL)
   {
      head = new_node;
      return;
   }
   //traverse till the end
   else{
      while (last->next != NULL)
      {
         last = last->next;
      }
      last->next = new_node;
   }
}
```

```
void insert_at_pos(int n)
{
    int pos;
    printf("Enter the position\n");
    scanf("%d",&pos);
    struct Node* ptr =(struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = head;
    ptr->data = n;

    if(pos==1){
        ptr->next = temp;
        head = ptr;
    }

    else{
        for(int i=1;i<pos-1 && temp!=NULL ;i++){
            temp = temp->next;
        }
        ptr->next = temp->next ;
        temp->next = ptr;
    }
}

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

Output:



C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\insertion_sinc

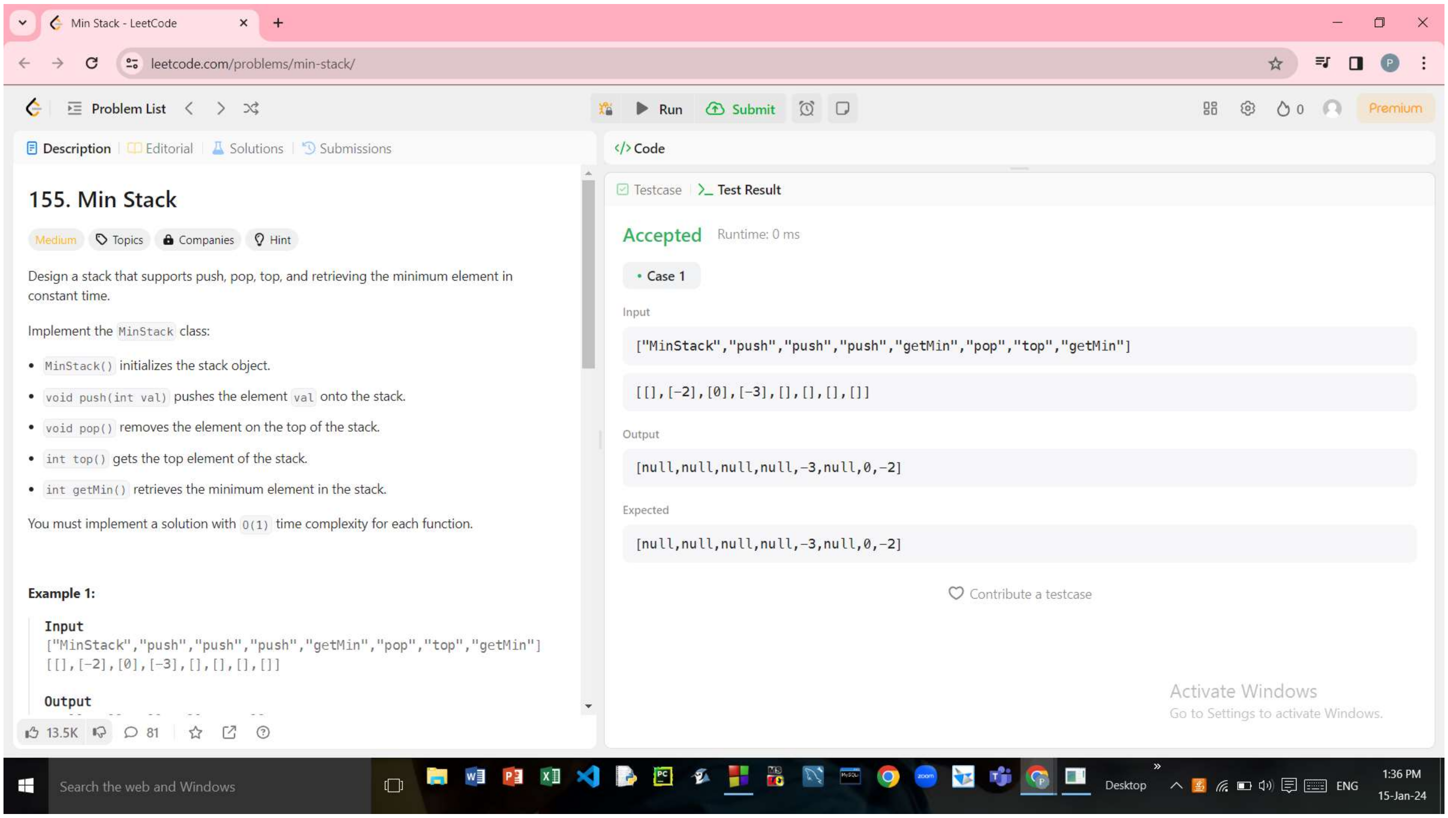
```
1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit
Enter your choice: 1
Enter the insert element
10
1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit
Enter your choice: 2
Enter the insert element
20
1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit
Enter your choice: 2
Enter the insert element
30
1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit
Enter your choice: 3
Enter the insert element
2
Enter the position
3
1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit
Enter your choice: 4
10
20
2
30
1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit
```

Program - Leetcode platform(Minimum Stack):

```
#include<stdio.h>
#include<stdlib.h>
#define max 1000

typedef struct {
    int top;
    int st[max];
    int min[max];
} MinStack;

MinStack* minStackCreate() {
    MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
    stack->top = -1;
    return stack;
}

void minStackPush(MinStack* obj, int val) {
    if(obj->top == max-1){
        printf("Stack Full\n");
        return;
    }
    obj->st[++obj->top] = val;

    if(obj->top > 0)
    {
        if(obj->min[obj->top - 1] < val)
            obj->min[obj->top] = obj->min[obj->top - 1];
        else
```

```
            obj->min[obj->top] = val;
    }
    else
        obj->min[obj->top] = val;
}

void minStackPop(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return;
    }
    else {
        obj->top -= 1;
    }
}

int minStackTop(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return -1;
    }
    return obj->st[obj->top];
}

int minStackGetMin(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("min Stack empty\n");
        return -1;
    }
    return obj->min[obj->top];
}

void minStackFree(MinStack* obj) {
    free(obj);
}

int main() {
    MinStack* obj = minStackCreate();

    minStackPush(obj, 3);
    minStackPush(obj, 5);
    minStackPush(obj, 2);
    minStackPush(obj, 1);

    printf("Min: %d\n", minStackGetMin(obj));



    printf("Top: %d\n", minStackTop(obj));

    minStackPop(obj);
    printf("Min: %d\n", minStackGetMin(obj));
    minStackFree(obj);

    return 0;
}
```

Output:



155. Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

```
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
```

Accepted Runtime: 0 ms

Case 1

Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
```

Output

```
[null,null,null,null,-3,null,0,-2]
```

Expected

```
[null,null,null,null,-3,null,0,-2]
```

Contribute a testcase

Activate Windows
Go to Settings to activate Windows.

Lab Program 5:

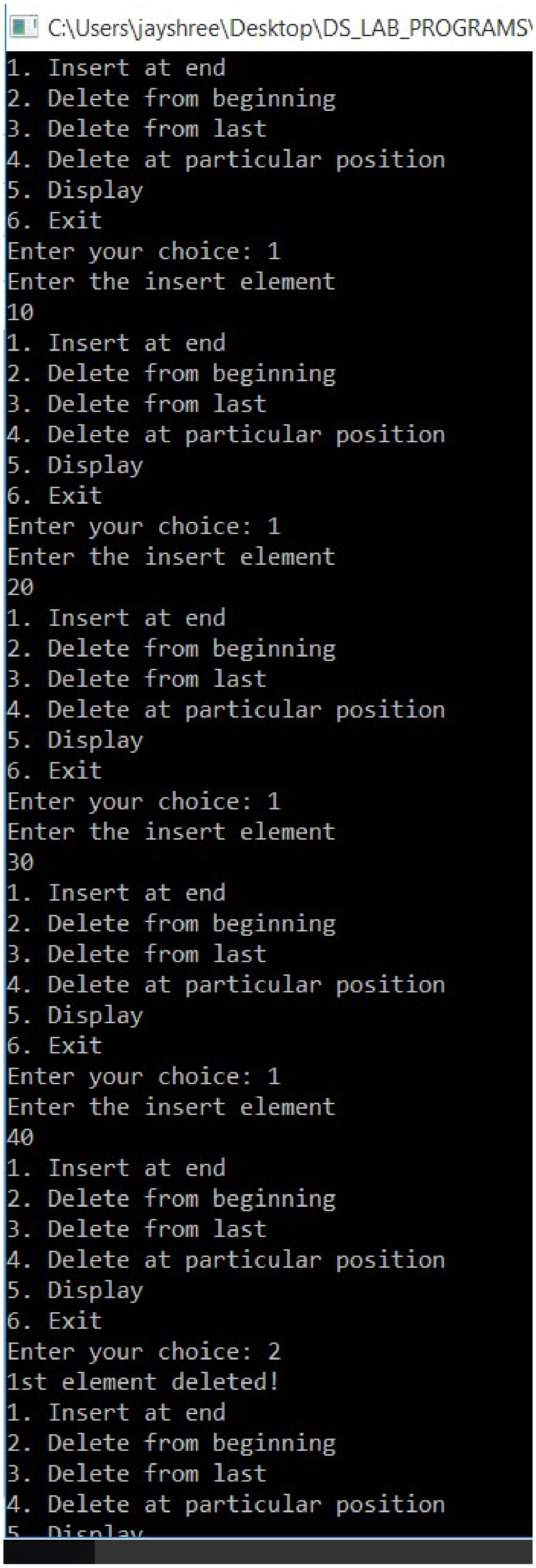
Write a Program to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

```
#include<stdio.h>

void append(int);

void fpop();

void lpop();

void mpop();

struct Node

{

   int data;

   struct Node *next;

};


struct Node *head = NULL;

int choice,pos,n;


void main()

{

   while(1){

   printf("1. Insert at end\n");

   printf("2. Delete from beginning\n");

   printf("3. Delete from last\n");

   printf("4. Delete at particular position\n");

   printf("5. Display\n");

   printf("6. Exit\n");


   printf("Enter your choice: ");

   scanf("%d",&choice);
```

```
    switch(choice)
    {
    case 1:
        printf("Enter the insert element\n");
        scanf("%d",&n);
        append(n);
        break;
    case 2:
        fpop();
        break;
    case 3:
        lpop();
        break;
    case 4:
        mpop();
        break;
    case 5:
        display();
        break;
    case 6:
        exit(0);
    default:
        printf("Enter correct choice\n");
        break;
        }
        continue;
    }
}
```

```
void append(int n)
{
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last  = head;  // if head is linked to one or more nodes already, then point last pointer to head.
    new_node->data = n;
    new_node->next = NULL;
    // if the list is empty.
    if(head == NULL)
    {
        head = new_node;
        return;
    }
    //traverse till the end
    else{
        while (last->next != NULL)
        {
            last = last->next;
        }
        last->next = new_node;
    }
}


void fpop()
{
struct Node *ptr;
   if(head == NULL)
        printf("List is empty\n");
   else
   {
    ptr = head;
    head = head->next;
    free(ptr);
    printf("1st element deleted!\n");
   }
}

void lpop()
{
    struct Node *ptr,*p1;

    if(head == NULL)
        printf("List is empty\n");

    else
    {
        if(head->next == NULL)
        {
        free(head);
        head = NULL;
        }

        else
        {   ptr = head;
            while(ptr->next != NULL)
            {
                p1 = ptr;
                ptr = ptr->next;
            }
            p1->next = NULL;
            free(ptr);
        }
        printf("Last element deleted!\n");
    }

}




void mpop()
{
    int pos;
    printf("Enter the position to be deleted\n");
    scanf("%d",&pos);
    struct Node *ptr,*ptr1;
    if(pos == 1)
    {
        ptr = head;
        free(ptr);
        head = NULL;
    }
    else
    {
        ptr = head;
        while(pos-1 != 0)
        {
            ptr1 = ptr;
            ptr = ptr->next;
            pos--;
        }
        ptr1->next = ptr->next;
        free(ptr);
    }
    printf("element deleted!\n");
}
void display()
{
    struct Node* node = head;
    while(node != NULL)
        {
            printf("%d\n",node->data);
            node = node->next;
        }
}
```

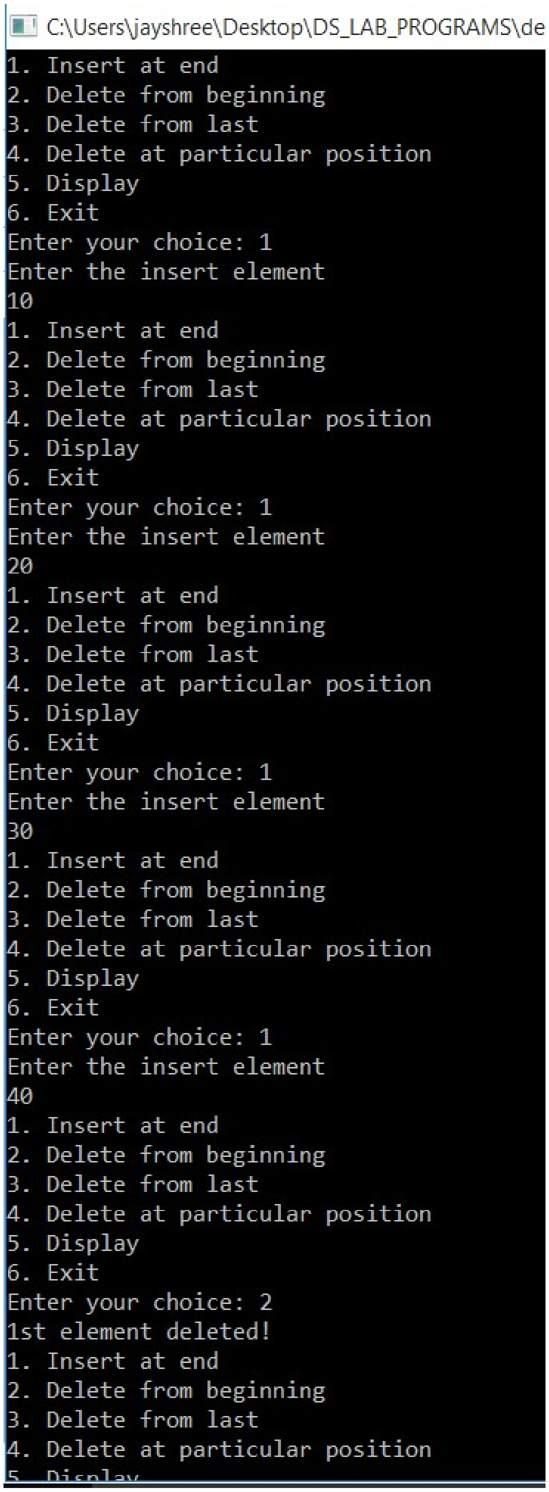
Output: 1)



C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\

```
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
10
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
20
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
30
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
40
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 2
1st element deleted!
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
```

2)



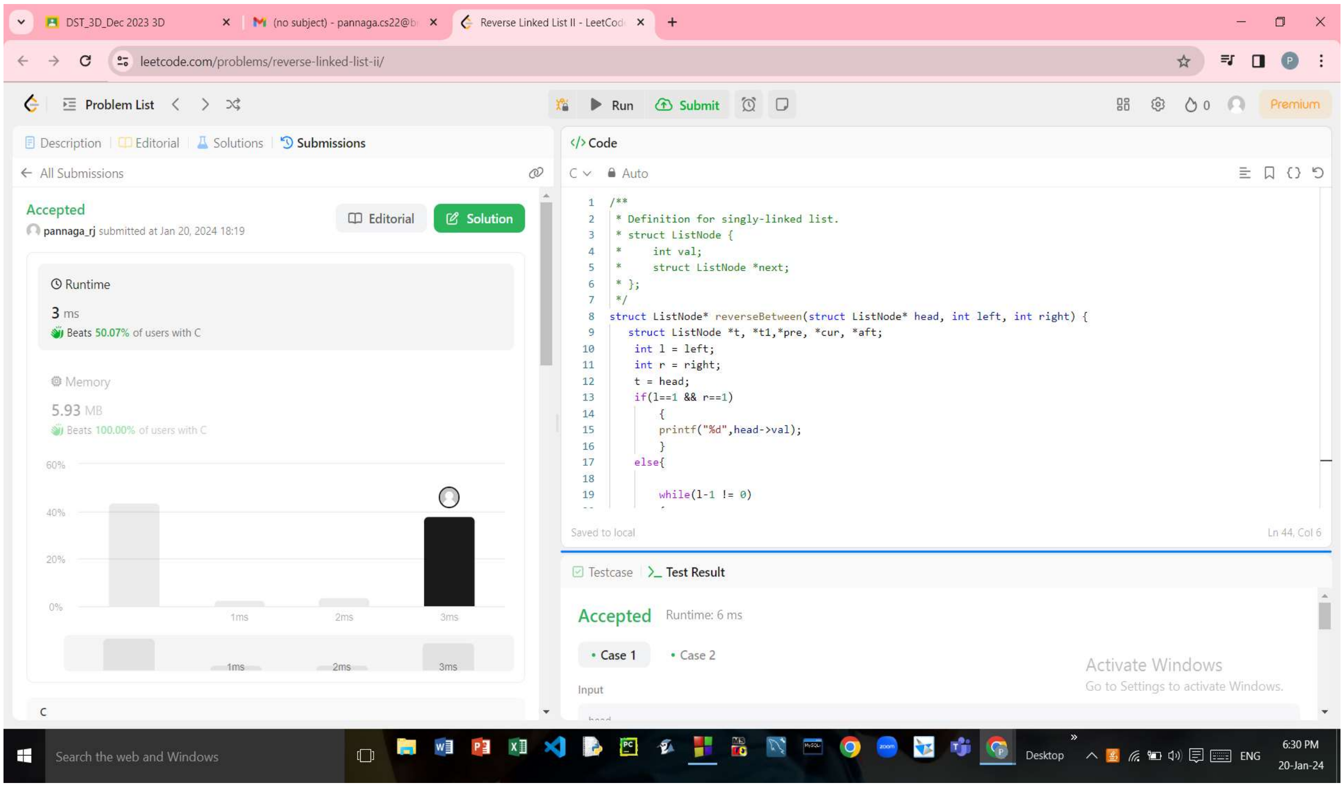
C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\de

```
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
10
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
20
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
30
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
40
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 2
1st element deleted!
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
```

Program - Leetcode platform (Reverse Linked List II):

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
   struct ListNode *t, *t1,*pre, *cur, *aft;
    int l = left;
    int r = right;
    t = head;
    if(l==1 && r==1)
        {
        printf("%d",head->val);
        }
    else{
        while(l-1 != 0)
        {
            t1 = t;
            t = t->next;
            l--;
        }
        pre = NULL;
        cur = aft = t;
        for(int i=left;i<=right;i++)
        {
            aft = aft->next;
            cur->next = pre;
            pre = cur;
            cur = aft;
        }
        t1->next = pre;
        t->next = aft;
    }
        if (left > 1) {
        return head;
    } else {
        return pre;
    }
}
```

Output:



leetcode.com/problems/reverse-linked-list-ii/

Accepted

pannaga_rj submitted at Jan 20, 2024 18:19

Runtime

3 ms

Beats 50.07% of users with C

Memory

5.93 MB

Beats 100.00% of users with C

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
   struct ListNode *t, *t1,*pre, *cur, *aft;
    int l = left;
    int r = right;
    t = head;
    if(l==1 && r==1)
        {
        printf("%d",head->val);
        }
    else{

        while(l-1 != 0)
```

Saved to local

Ln 44, Col 6

Accepted Runtime: 6 ms

Case 1 Case 2

Input

Lab Program 6

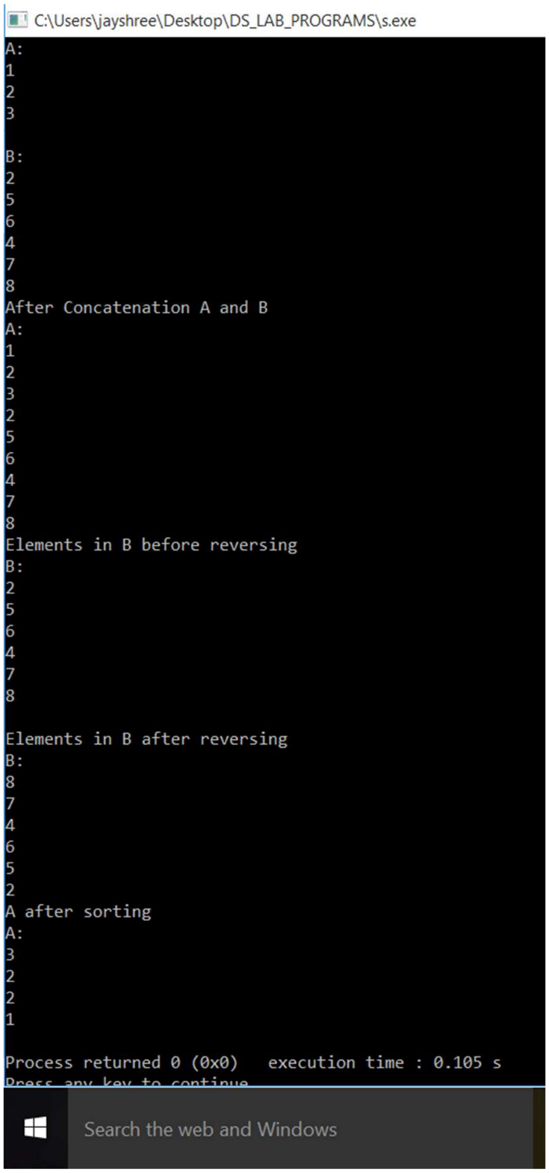
6a) Write a program to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```
#include<stdio.h>


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


struct node *head, *head1, *head3;



void create()
{
    head = (struct node*)malloc(sizeof(struct node));
    head1 = (struct node*)malloc(sizeof(struct node));
   struct node *last, *last1;
   int a[] = {1,2,3,4};
   int b[] = {2,5,6,4,7,8};

   head->data = a[0];
   head->next = NULL;

   head1->data = b[0];
   head1->next = NULL;

   last = head;
   last1 = head1;
```

```
  for(int i=1;i<sizeof(a)/sizeof(a[0]);i++)
  {
     struct node *t;
     t = (struct node*)malloc(sizeof(struct node));
     t->data = a[i];
     t->next = NULL;
     last->next = t;
     last = t;
  }

  for(int i=1;i<sizeof(b)/sizeof(b[0]);i++)
  {
      struct node *t1;
      t1 = (struct node*)malloc(sizeof(struct node));
      t1->data = b[i];
      t1->next = NULL;
      last1->next = t1;
      last1 = t1;
   }

}



void display_a()
{
   struct node *n = head;

   printf("A:\n");
   while(n!=NULL)
```

```
    {
        printf("%d\n",n->data);
        n = n->next;
    }
    return;
}

void display_b()
{
    struct node *n1 = head1;

    printf("B:\n");
    while(n1!=NULL)
    {
        printf("%d\n",n1->data);
        n1 = n1->next;
    }
}

void reverse()
{
    struct node *t, *pre, *cur, *aft;
    printf("Elements in B before reversing\n");
    display_b();
    t = head1;
    pre = NULL;
    cur = aft = t;
    while(cur != NULL)
    {
        aft = aft->next;
        cur->next = pre;
        pre = cur;
        cur = aft;
    }
    head1 = pre;
    printf("\n");
    printf("Elements in B after reversing\n");
    display_b();
}

void concat()
{
    struct node *n, *n1;
    if(head == NULL || head1 == NULL)
    {
        if(head == NULL)
            display_b();
        else
            display_a();
    }
    else
    {
        n=head;
        while(n!=NULL)
        {
            n1 = n;
            n = n->next;
        }
        n1->next = head1;
        display_a();
    }

}

void sort()
{
    struct node *p, *p1;
    for(p = head; p!= NULL; p=p->next)
    {
        for(p1 = p->next;p1!=NULL; p1=p1->next)
        {
            if(p->data < p1->data)
            {
                int temp = p->data;
                p->data = p1->data;
                p1->data = temp;
            }
        }
    }
    printf("A after sorting\n");
    display_a();
}

void main()
{
    create();
    display_a();
    printf("\n");
    display_b();
    printf("After Concatenation A and B\n");
    concat();
    reverse();
    sort();
}
```

Output:


C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\s.exe
Search the web and Windows

```
A:
1
2
3

B:
2
5
6
4
7
8
After Concatenation A and B
A:
1
2
3
2
5
6
4
7
8
Elements in B before reversing
B:
2
5
6
4
7
8

Elements in B after reversing
B:
8
7
4
6
5
2
A after sorting
A:
3
2
2
1

Process returned 0 (0x0)   execution time : 0.105 s
Press any key to continue.
```

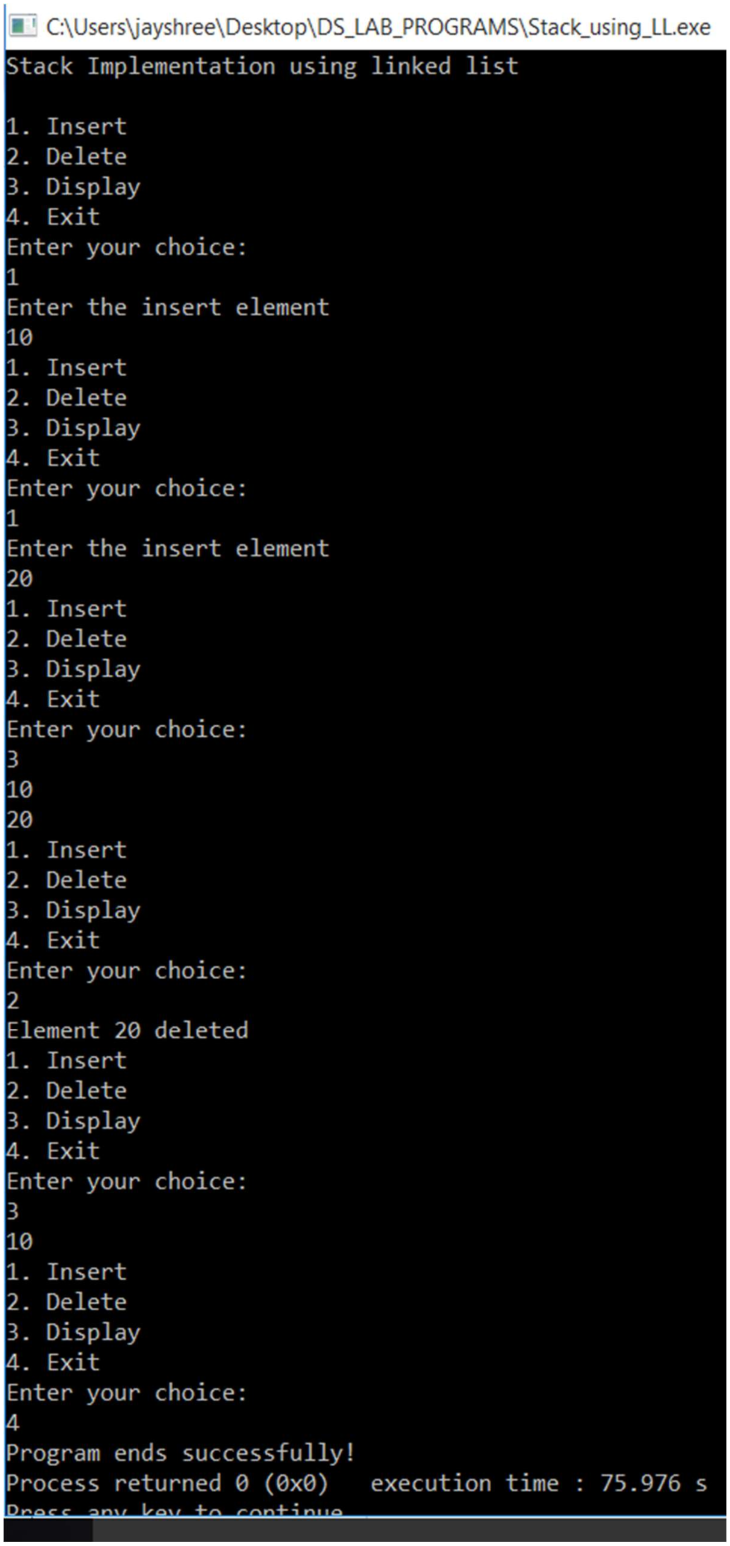
6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

Stack Implementation

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();

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

struct node *head = NULL;
void main()
{
    int ch;
    printf("Stack Implementation using linked list\n\n");
    while(1){
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice:\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Program ends successfully!");
                exit(0);
            default:
                printf("Enter a valid number...\n");
        }
        continue;
    }
}

void push()
{
    int n;
    printf("Enter the insert element\n");
    scanf("%d",&n);

    struct node *new_node = (struct node*)malloc(sizeof(struct node));
    new_node -> data = n;
    new_node->next = NULL;

    if(head == NULL)
        head = new_node;
    else
    {   struct node *p;
        p = head;
        while(p->next != NULL)
        {
            p = p->next;
        }
        p->next = new_node;
    }
    return;
}

void pop()
{
    struct node *ptr,*p2;
    if(head == NULL)
    {
        printf("List is empty\n");
        exit(0);
    }
    else
    {   if(head->next == NULL)
        {
            printf("Element %d deleted\n",head->data);
            free(head);
            head = NULL;
        }
        else
        {
            ptr = head;
            while(ptr->next != NULL)
            {
                p2 = ptr;
                ptr = ptr->next;
            }
            printf("Element %d deleted\n",ptr->data);
            p2->next = NULL;
            free(ptr);
        }
        return;
    }
}
void display()
{
    struct node *n;
    if(head == NULL)
    {
        printf("List is empty");
        exit(0);
    }
    else{
    n = head;
    while(n != NULL)
        {
            printf("%d\n",n->data);
            n = n->next;
        }
    }
```

```
}
```

Output:


C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Stack_using_LL.exe

```
Stack Implementation using linked list

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the insert element
10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the insert element
20
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
3
10
20
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
2
Element 20 deleted
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
3
10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
4
Program ends successfully!
Process returned 0 (0x0)   execution time : 75.976 s
Press any key to continue.
```

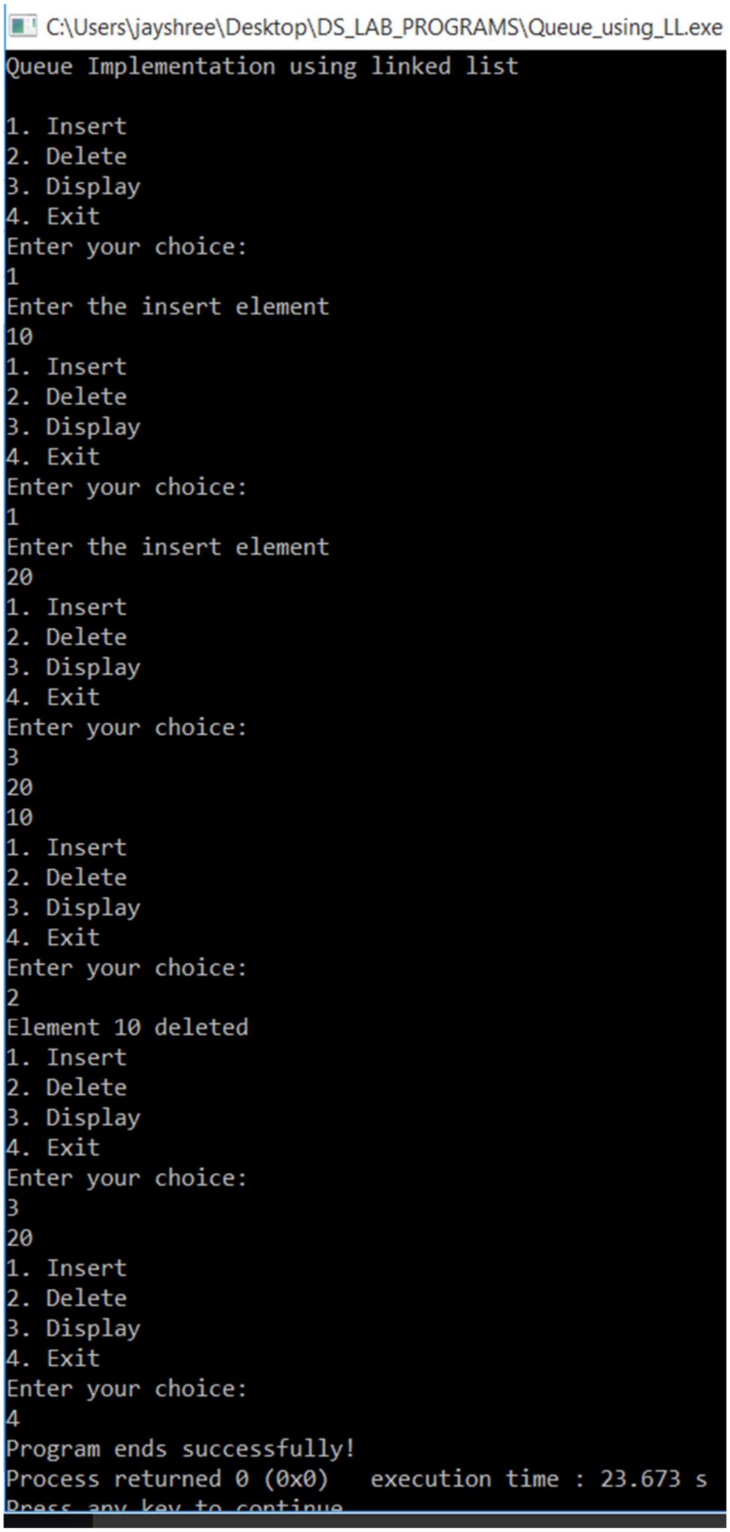
Queue Implementation

```
#include<stdio.h>

#include<stdlib.h>

void enqueue();

void dequeue();

void display();


struct node{

   int data;

   struct node *next;

};


struct node *head = NULL;

void main()

{

   int ch;

   printf("Queue Implementation using linked list\n\n");

   while(1){

      printf("1. Insert\n");

      printf("2. Delete\n");

      printf("3. Display\n");

      printf("4. Exit\n");

      printf("Enter your choice:\n");

      scanf("%d",&ch);

      switch(ch)

      {

         case 1:

            enqueue();

            break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Program ends successfully!");
                exit(0);
            default:
                printf("Enter a valid number...\n");
        }
        continue;
    }
}

void enqueue()
{
    int n;
    printf("Enter the insert element\n");
    scanf("%d",&n);

    struct node *new_node =(struct node*)malloc(sizeof(struct node));
    new_node->data = n;
    new_node->next = head;
    head = new_node;
}

void dequeue()
```

```
{
   if(head == NULL)
   {
      printf("List is empty");
      exit(0);
   }
   else
   {
      if(head->next == NULL)
      {
         printf("Element %d deleted\n",head->data);
         free(head);
         head = NULL;
      }
      else
      {
         struct node *p,*p1;
         p = head;
         while(p->next != NULL)
         {
            p1 = p;
            p = p->next;
         }
         p1->next = NULL;
         printf("Element %d deleted\n",p->data);
         free(p);
      }
   }
}
```

```
void display()
{
    struct node *n;
    if(head == NULL)
    {
        printf("List is empty");
        exit(0);
    }
    else{
    n = head;
    while(n != NULL)
        {
            printf("%d\n",n->data);
            n = n->next;
        }
    }
}
```

Output:


C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Queue_using_LL.exe

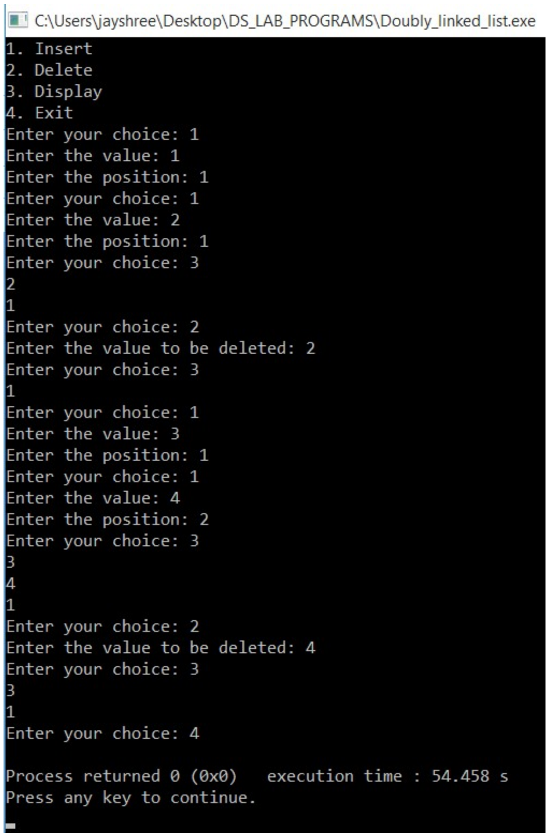
```
Queue Implementation using linked list

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the insert element
10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the insert element
20
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
3
20
10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
2
Element 10 deleted
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
3
20
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
4
Program ends successfully!
Process returned 0 (0x0)   execution time : 23.673 s
Press any key to continue
```

Lab Program 7:

WAP to Implement doubly link list with primitive operations. a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value. d) Display the contents of the list

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

struct node
{
    struct node *pre;
    struct node *next;
    int data;
};
struct node *head = NULL;

void insert_left();
void delete_val();
void display();

int main()
{
    int ch;
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Display\n");
    printf("4. Exit\n");

    while (1)
    {
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insert_left();
            break;
        case 2:
            delete_val();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Enter correct choice:\n");
            break;
        }
    }
}

void insert_left()
{
    int val, pos;
    printf("Enter the value: ");
    scanf("%d", &val);
    printf("Enter the position: ");
    scanf("%d", &pos);
    struct node *new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = val;

    if (pos == 1)
    {
        new_node->pre = NULL;
        new_node->next = head;
        if (head != NULL)
        {
            head->pre = new_node;  // Set the "pre" pointer of the current head to the new node, if it has one node already.
        }
        head = new_node;
    }
    else
    {
        struct node *t;
        if(pos>1)
        {
            t = head;
            while(pos-1 != 0)
            {
                t = t->next;
                pos--;
            }
            new_node->pre = t->pre;
            new_node->next = t;
            t->pre->next = new_node;
            t->pre = new_node;
        }
    }
}

void delete_val()
{
    int d, c = 1;
    printf("Enter the value to be deleted: ");
    scanf("%d", &d);

    struct node *t = head;

    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    else
    {
        while(t != NULL)
        {
            if(t->data == d)
            {
                break;
            }
            t = t->next;
            c++;
        }
        if(t != NULL)
        {
            if(c==1)
            {
                head = t->next;

                if(head != NULL)
                    t->next->pre = NULL;

                free(t);
            }
            else
            {
            t->pre->next = t->next;

            if( t->next != NULL)
                t->next->pre = t->pre;

            free(t);
            }
        }
        else
        {
            printf("Element not found\n");
        }
    }
}
void display()
{
    struct node *t = head;
    if (head == NULL){
        printf("List is empty\n");
        return;
    }
    else{
        while (t != NULL)
        {
            printf("%d\n", t->data);
            t = t->next;
        }
    }
}
```

Output:

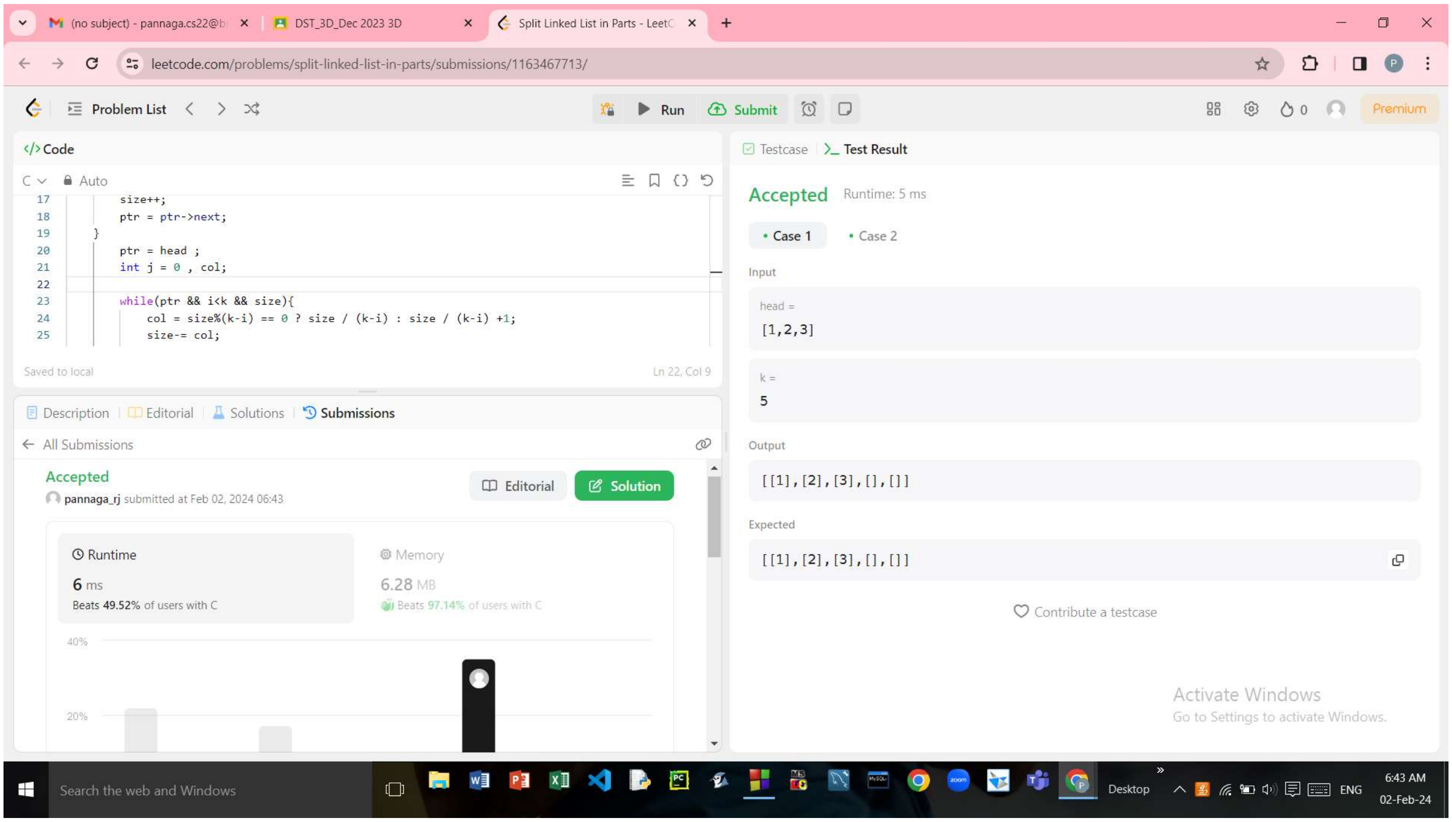

C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Doubly_linked_list.exe

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value: 1
Enter the position: 1
Enter your choice: 1
Enter the value: 2
Enter the position: 1
Enter your choice: 3
2
1
Enter your choice: 2
Enter the value to be deleted: 2
Enter your choice: 3
1
Enter your choice: 1
Enter the value: 3
Enter the position: 1
Enter your choice: 1
Enter the value: 4
Enter the position: 2
Enter your choice: 3
3
4
1
Enter your choice: 2
Enter the value to be deleted: 4
Enter your choice: 3
3
1
Enter your choice: 4

Process returned 0 (0x0)   execution time : 54.458 s
Press any key to continue.
```

Program - Leetcode platform(Split Linked List into parts):

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize){
    struct ListNode **arr = (struct ListNode*)calloc(k+1,sizeof(struct ListNode));
    int size = 0 ;
    struct ListNode *ptr = head , *next ;
    *returnSize = k; int i = 0;
    while(ptr){
        size++;
        ptr = ptr->next;
    }
        ptr = head ;
        int j = 0 , col;

        while(ptr && i<k && size){
            col = size%(k-i) == 0 ? size / (k-i) : size / (k-i) +1;
            size-= col;
            head = ptr;
            j = 0;
            while(ptr && j<col-1){
                ptr= ptr->next;
                j++;
            }
            next = ptr->next;
            arr[i++] = head , ptr->next = NULL;
            ptr = next;
        }
    return arr;
}
```

Output:



leetcode.com/problems/split-linked-list-in-parts/submissions/1163467713/

```
        size++;
        ptr = ptr->next;
    }

        ptr = head ;
        int j = 0 , col;

        while(ptr && i<k && size){
            col = size%(k-i) == 0 ? size / (k-i) : size / (k-i) +1;
            size-= col;
```

Accepted

pannaga_rj submitted at Feb 02, 2024 06:43

Runtime 6 ms Beats 49.52% of users with C

Memory 6.28 MB Beats 97.14% of users with C

Accepted Runtime: 5 ms

Case 1 Case 2

Input

head =
[1,2,3]

k =
5

Output

```
[[1],[2],[3],[],[]]
```

Expected

```
[[1],[2],[3],[],[]]
```

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 node

{

   int data;

   struct node *l;

   struct node *r;

};


struct node *root = NULL;


struct node *create(int d)

{

   struct node *t = (struct node*)malloc(sizeof(struct node));

   t->data = d;

   t->l = t->r = NULL;

   return t;

}

struct node *insert(struct node* t, int val)

{

   if(t == NULL)

   {

      return create(val);

   }

   if(val < t->data)
```

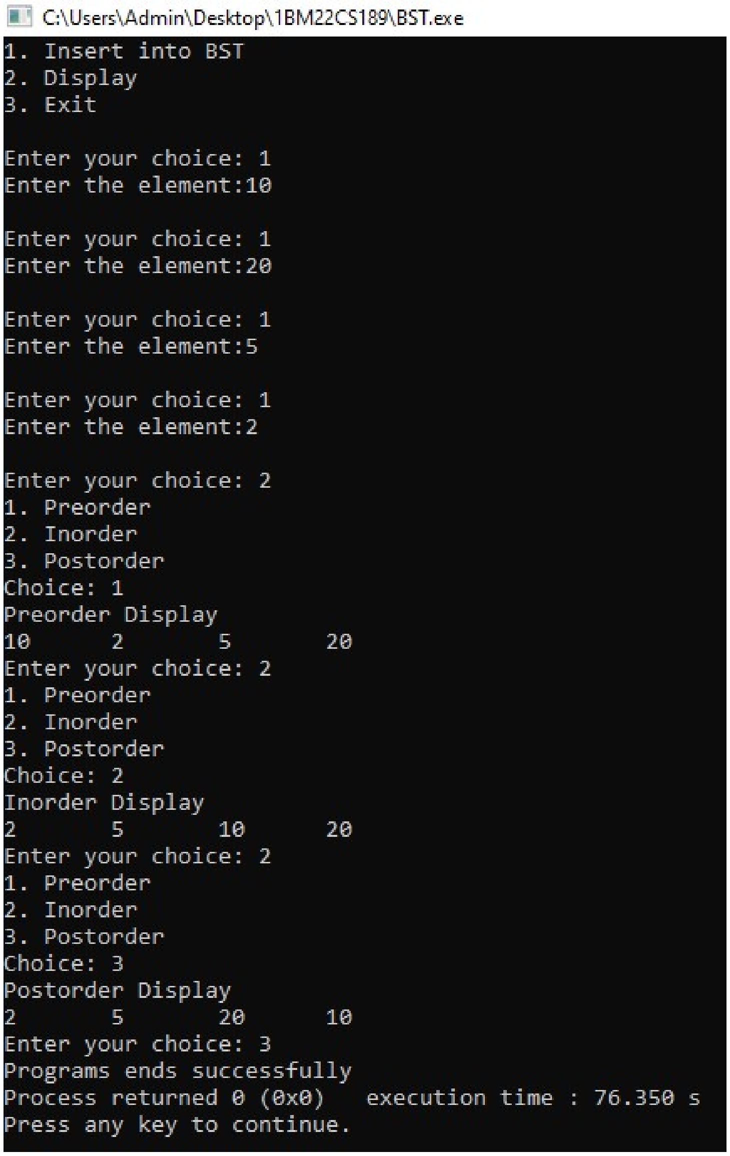
```
    {
        t->l = insert(t->l, val);
    }
    else if(val > t->data)
    {
        t->r = insert(t->r, val);
    }
    return t;
}

struct node* inorder(struct node* root)
{
    if(root != NULL)
    {
        inorder(root->l);
        printf("%d\t",(root->data));
        inorder(root->r);
    }
}

struct node* postorder(struct node* root)
{
    if(root != NULL)
    {
        postorder(root->l);
        postorder(root->r);
        printf("%d\t",root->data);
    }
}
```

```
struct node* preorder(struct node* root)
{
   if(root != NULL)
   {
      printf("%d\t",root->data);
      postorder(root->l);
      postorder(root->r);

   }
}

void main()
{
   int ch, d, d1;
   printf("1. Insert into BST\n");
   printf("2. Display\n");
   printf("3. Exit\n");
   while(1)
   {
      printf("\nEnter your choice: ");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         printf("Enter the element:");
         scanf("%d",&d);
         root = insert(root, d);
         break;
```

```
case 2:
   printf("1. Preorder\n");
   printf("2. Inorder\n");
   printf("3. Postorder\n");
   printf("Choice: ");
   scanf("%d",&d1);
   switch(d1)
   {
   case 1:
      printf("Preorder Display\n");
      preorder(root);
      break;
   case 2:
      printf("Inorder Display\n");
      inorder(root);
      break;
   case 3:
      printf("Postorder Display\n");
      postorder(root);
      break;
   default:
      printf("Enter correct choice.");
      break;
   }
   break;

case 3:
   printf("Programs ends successfully");
   exit(0);
```

```
        default:
            printf("Enter correct choice");
            break;
        }
    }
}
```

Output:



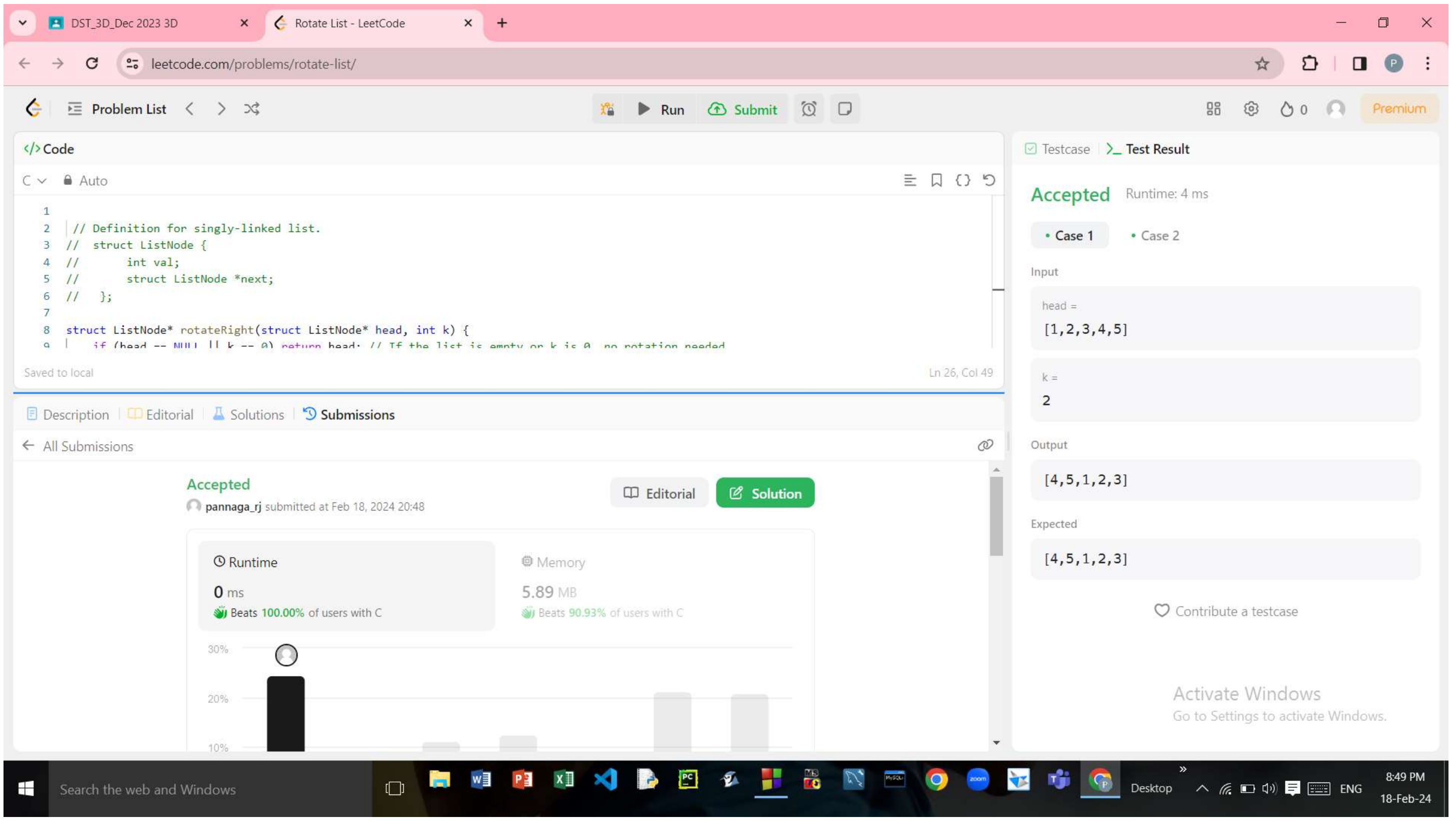
C:\Users\Admin\Desktop\1BM22CS189\BST.exe

```
1. Insert into BST
2. Display
3. Exit

Enter your choice: 1
Enter the element:10

Enter your choice: 1
Enter the element:20

Enter your choice: 1
Enter the element:5

Enter your choice: 1
Enter the element:2

Enter your choice: 2
1. Preorder
2. Inorder
3. Postorder
Choice: 1
Preorder Display
10      2       5       20
Enter your choice: 2
1. Preorder
2. Inorder
3. Postorder
Choice: 2
Inorder Display
2       5       10      20
Enter your choice: 2
1. Preorder
2. Inorder
3. Postorder
Choice: 3
Postorder Display
2       5       20      10
Enter your choice: 3
Programs ends successfully
Process returned 0 (0x0)   execution time : 76.350 s
Press any key to continue.
```

Program - Leetcode platform(Rotate Linked list):

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

struct ListNode* rotateRight(struct ListNode* head, int k) {
    if (head == NULL || k == 0) return head; // If the list is empty or k is 0, no rotation needed

    struct ListNode *t = head;
    int len = 1; // Length of the list

    // length of the list
    while (t->next != NULL) {
        t = t->next;
        len++;
    }

    // Adjust k if it's greater than the length of the list
    k = k % len;
    if (k == 0) return head; // no rotation needed

    t->next = head; // Connect last to first node (circular)

    // Traverse again to find the new last node
    t = head;
    for (int i = 0; i < len - k - 1; i++) {
        t = t->next;
    }
    // New head after rotation
    struct ListNode *newHead = t->next;
    t->next = NULL; // Break the circle
    return newHead;
}
```

Output:



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

struct ListNode* rotateRight(struct ListNode* head, int k) {
    if (head == NULL || k == 0) return head; // If the list is empty or k is 0, no rotation needed
```

Saved to local

Ln 26, Col 49

Accepted

pannaga_rj submitted at Feb 18, 2024 20:48

Runtime

0 ms

Beats 100.00% of users with C

Memory

5.89 MB

Beats 90.93% of users with C

Accepted Runtime: 4 ms

Input

head =
[1,2,3,4,5]

k =
2

Output

[4,5,1,2,3]

Expected

[4,5,1,2,3]

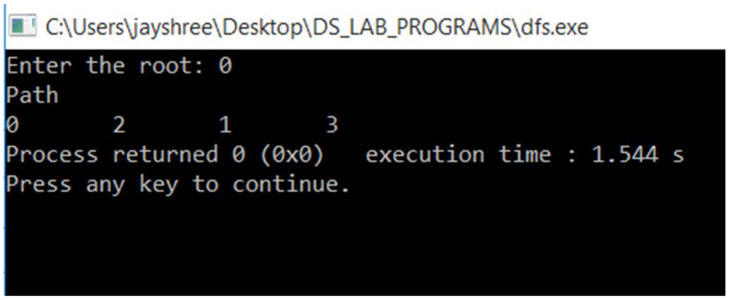
Lab Program 9:

9a) Write a program to traverse a graph using BFS method.

```
#include <stdio.h>

#define r 4

int fr = 0;
int rear = 0;
int v[r];
int a[r][r] = {{1,0,1,0}, {0,0,1,1}, {0,1,0,1}, {1,1,1,1}};


int q[r];

void bfs(int n)
{
    int roo;
    while (fr < rear) {
        roo = q[fr++];
        printf("%d\t", roo);
        for (int j = 0; j < r; j++) {
            if (a[roo][j] && !v[j]) {
                v[j] = 1;
                q[rear++] = j;
            }
        }
    }
}
```

```
int main()
{
    int root;
    for (int i = 0; i < r; i++)
    {
        v[i] = 0;
    }
    printf("Enter the root: ");
    scanf("%d", &root);
    q[rear++] = root;
    v[root] = 1;
    bfs(root);

    return 0;
}
```

Output:

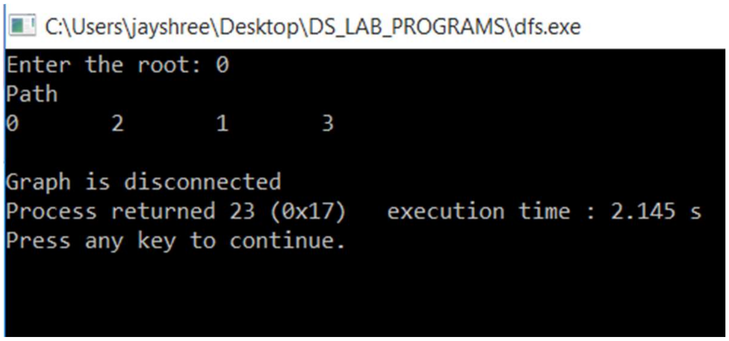


```
Enter the root: 0
Path
0       2       1       3
Process returned 0 (0x0)   execution time : 1.544 s
Press any key to continue.
```

9b) Write a program to check whether given graph is connected or not using DFS method.

```
// Depth wise Search
#include<stdio.h>

# define r 5

int a[r][r] = {{1,0,1,0}, {0,0,1,1}, {0,1,0,1}, {1,1,1,1}};
int v[r];

void dfs(int root)
{

    printf("%d\t",root);
    v[root] = 1;
    for(int i=0;i<r;i++)
    {
        if(a[root][i] && !v[i])
        {
            dfs(i);
        }
    }
}

void connected()
{
    int flag = 0;
    for(int i=0;i<r;i++)
    {
        if(v[i]==0)
            flag = 1;
    }

    if(flag==1)
        printf("\n\nGraph is disconnected");
    else
        printf("\n\nGraph is connected");

}
void main()
{
    int  root;
    for(int i=0;i<r;i++)
    {
        v[i] = 0;
    }

    printf("Enter the root: ");
    scanf("%d",&root);
    printf("Path\n");
    dfs(root);
    connected();
}
```

Output:



C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\dfs.exe

```
Enter the root: 0
Path
0       2       1       3

Graph is disconnected
Process returned 23 (0x17)   execution time : 2.145 s
Press any key to continue.
```

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.

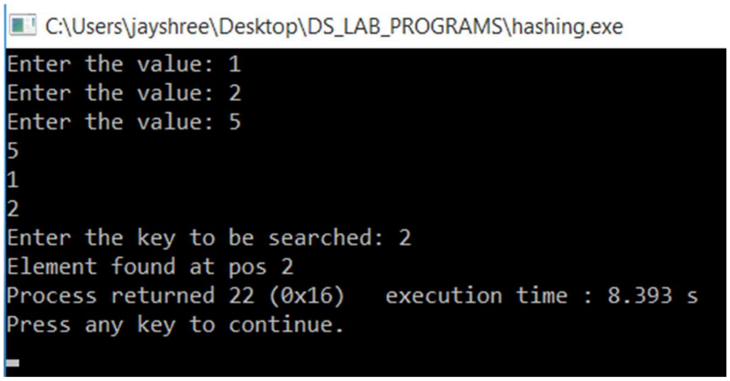
Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L.

Resolve the collision (if any) using linear probing.

```
#include<stdio.h>

# define ts 3

int hash_array[ts];

void hashing(int key)
{
   int hkey, index;
   int i = 0;
```

```
    hkey = key % ts;
    while(i<ts)
    {
        index = (hkey+i) % ts;
        if(hash_array[index] == -1)
        {
            hash_array[index] = key;
            break;
        }
        i = i+1;
    }
}


void print()
{
    for(int i=0;i<ts;i++)
    {
        printf("%d\n",hash_array[i]);
    }
}

void search(int s)
{
    int hkey = s % ts;
    int i = 0, flag = 0;
    int index;
    while(i < ts)
    {
        index = (hkey+i) % ts;
        if(hash_array[index] == s)
        {
            flag = 1;
            printf("Element found at pos %d",index);
            break;
        }
        i+=1;
    }

    if(flag == 0)
        printf("Not found");

}

void main()
{
    int key, s;
    for(int j=0;j<ts;j++)
    {
        hash_array[j] = -1;
    }

    for(int c = 0;c<ts;c++)
    {
        printf("Enter the value: ");
        scanf("%d",&key);
        hashing(key);
    }
    printf("Enter the key to be searched: ");
    scanf("%d", &s);
    search(s);
}
```

Output:



```
Enter the value: 1
Enter the value: 2
Enter the value: 5
5
1
2
Enter the key to be searched: 2
Element found at pos 2
Process returned 22 (0x16)   execution time : 8.393 s
Press any key to continue.
```