VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

PRAKRUTHI B S (2023BMS02609)

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 PRAKRUTHI B S **(2023BMS02609)**, 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.

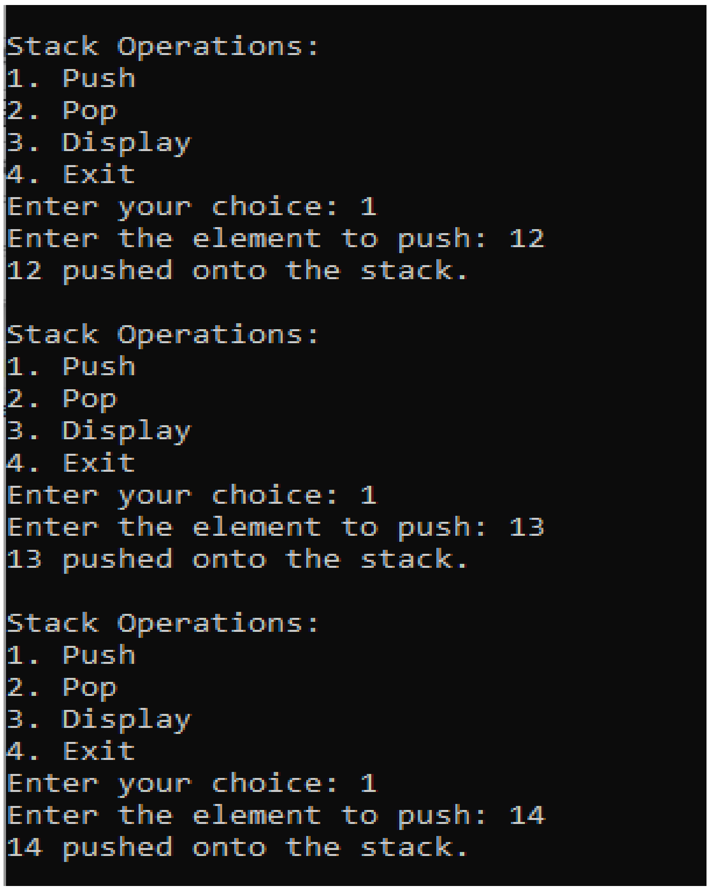
Lab program 1:

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.

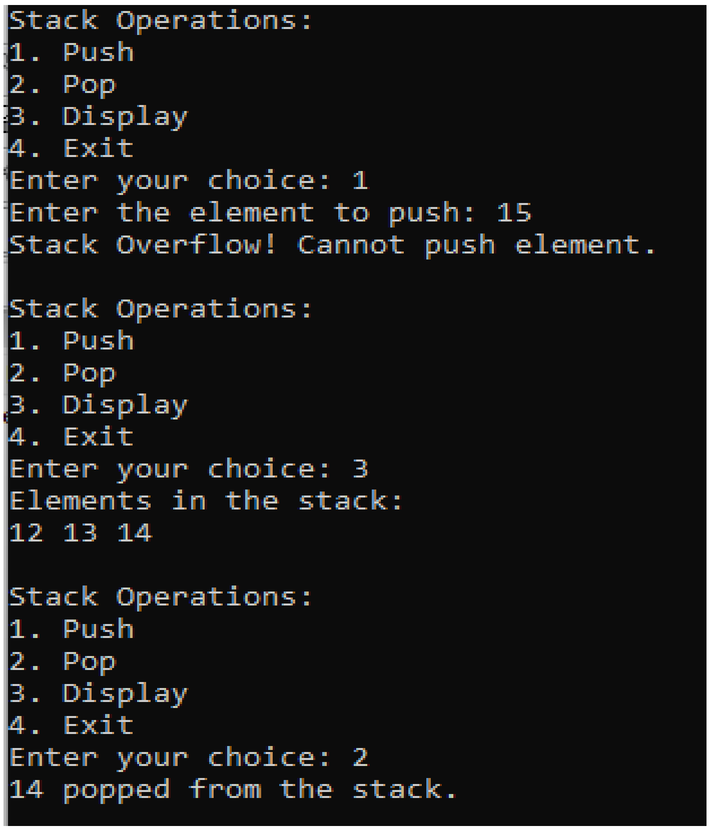
```
#include<stdio.h>

#include<stdlib.h>

#define STACK_SIZE 3

int stack[STACK_SIZE];

int top = -1;

void push(int element);

void pop();

void display();


int main() {

    int choice, element;

    do {

53

        printf("\nStack Operations:\n");

        printf("1. Push\n");

        printf("2. Pop\n");

        printf("3. Display\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);



        switch (choice) {
            case 1:
                printf("Enter the element to push: ");
                scanf("%d", &element);
                push(element);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 4);

    return 0;
}
void push(int element) {
    if (top == STACK_SIZE - 1) {
        printf("Stack Overflow! Cannot push element.\n");
    } else {
        top++;
        stack[top] = element;
        printf("%d pushed onto the stack.\n", element);
    }
```

```
}
void pop() {
    if (top == -1) {
        printf("Stack Underflow! Cannot pop element.\n");
    } else {
        printf("%d popped from the stack.\n", stack[top]);
        top--;
    }
}
void display() {
    if (top == -1) {
        printf("Stack is empty. Nothing to display.\n");
    } else {
        printf("Elements in the stack:\n");
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}
```

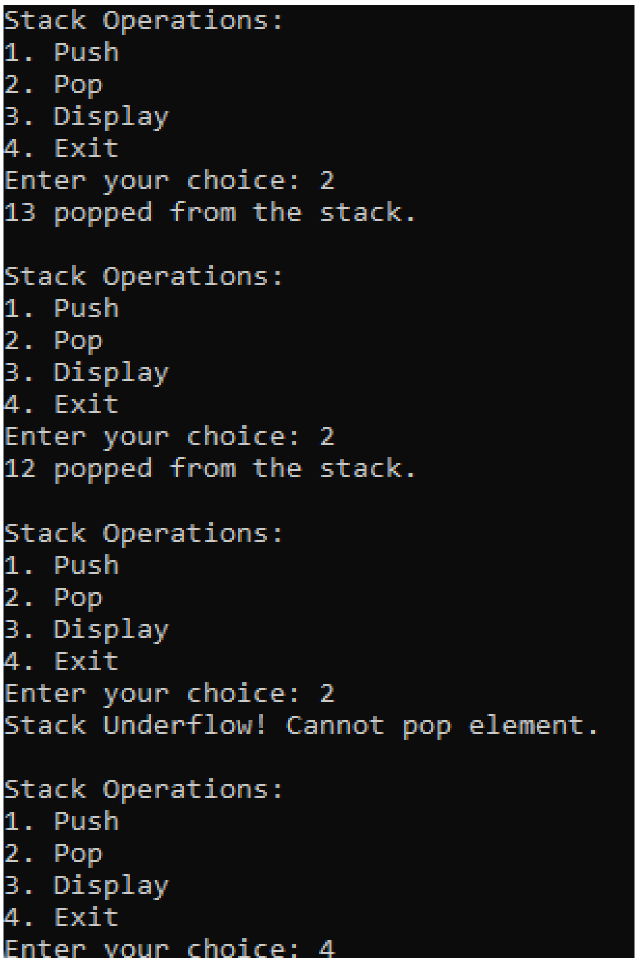
OUTPUT:



```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 12
12 pushed onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 13
13 pushed onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 14
14 pushed onto the stack.
```



```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 15
Stack Overflow! Cannot push element.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Elements in the stack:
12 13 14

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
14 popped from the stack.
```



```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
13 popped from the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
12 popped from the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Stack Underflow! Cannot pop element.

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

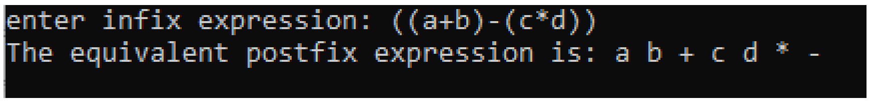
Lab Program 2

2a)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<stdlib.h>

#include<string.h>

#define MAX 100


char stack[MAX];

char infix[MAX];

char postfix[MAX];

int top=-1;


void push(char);

char pop();

int isEmpty();

void inToPost();

void print();

int precedence(char);


int main()

{

    printf("enter infix expression: ");

    gets(infix);

    inToPost();

    print();

    return 0;

}
```

```
void inToPost()
{
    int i,j=0;
    char symbol,next;
    for(i=0;i<strlen(infix);i++)
    {
        symbol=infix[i];
        switch(symbol)
        {
        case '(':
            push(symbol);
            break;
        case ')':
            while((next=pop())!='(')
                    postfix[j++]=next;
                    break;
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while (!isEmpty() && precedence(stack[top]) >= precedence(symbol))
                postfix[j++] = pop();
            push(symbol);
            break;
        default:
            postfix[j++] = symbol;
        }
    }
    while (!isEmpty())
        postfix[j++] = pop();
    postfix[j] = '\0';
}
int precedence(char symbol)
{
    switch (symbol)
    {
    case '^':
        return 3;
    case '/':
    case '*':
        return 2;
    case '+':
    case '-':
        return 1;
    default:
        return 0;
    }
}

void print()
{
    int i = 0;
    printf("The equivalent postfix expression is: ");
    while (postfix[i])
    {
        printf("%c ", postfix[i++]);
    }
    printf("\n");
}
void push(char c)
{
    if(top==MAX-1)
    {
        printf("stack overflow");
        return;
    }
    top++;
    stack[top]=c;
}
char pop()
{
    char c;
    if(top==-1)
    {
        printf("stack underflow");
        exit(1);
    }
    c=stack[top];
    top=top-1;
    return c;
}
int isEmpty()
{
    if(top==-1)
        return 1;
    else
        return 0;
}
```

OUTPUT:



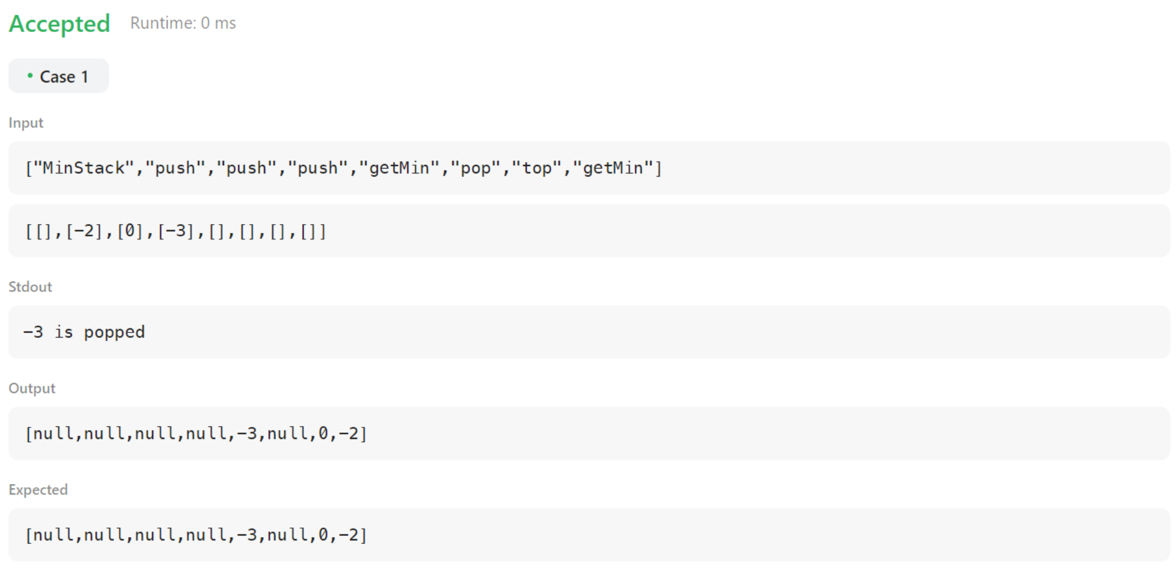
```
enter infix expression: ((a+b)-(c*d))
The equivalent postfix expression is: a b + c d * -
```

2b)Program - Leetcode platform- 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.

```
typedef struct {
    int size;
    int top;
    int *s;
    int *minstack;

} MinStack;


MinStack* minStackCreate() {
    MinStack *st=(MinStack*) malloc(sizeof(MinStack));
    if(st==NULL)
    {
        printf("memory alloction failed");
        exit(0);
    }
    st->size=1000;
    st->top=-1;
    st->s=(int*) malloc (st->size*sizeof(int));
    st->minstack = (int*) malloc (st->size * sizeof(int));
    if(st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
    return st;

}

void minStackPush(MinStack* obj, int val) {
    if(obj->top==obj->size-1)
    {
        printf("stack is overflow");

    }
    else{
        obj->top++;
        obj->s[obj->top]=val;
        if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
            obj->minstack[obj->top] = val;
        } else {
            obj->minstack[obj->top] = obj->minstack[obj->top - 1];
        }
    }


}

void minStackPop(MinStack* obj) {
    int value;
    if(obj->top==-1)
    {
        printf("underflow");

    }
    else
    {
        value=obj->s[obj->top];
        obj->top--;
        printf("%d is popped\n",value);
    }

}

int minStackTop(MinStack* obj) {
    int value=-1;
    if(obj->top==-1)
    {
        printf("underflow\n");
        exit(0);

    }
    else
    {
        value=obj->s[obj->top];
        return value;
     }
}

int minStackGetMin(MinStack* obj) {
    if(obj->top==-1)
    {
        printf("underflow\n");
        exit(0);

    }
    else
    {
        return obj->minstack[obj->top];
    }
}
void minStackFree(MinStack* obj) {
    free(obj->s);
    free(obj->minstack);
    free(obj);
}
/**
 * Your MinStack struct will be instantiated and called as such:
 * MinStack* obj = minStackCreate();
 * minStackPush(obj, val);

 * minStackPop(obj);

 * int param_3 = minStackTop(obj);

 * int param_4 = minStackGetMin(obj);

 * minStackFree(obj);
*/
```

OUTPUT:



Accepted Runtime: 0 ms

• Case 1

Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
```

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

Stdout

```
-3 is popped
```

Output

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

Expected

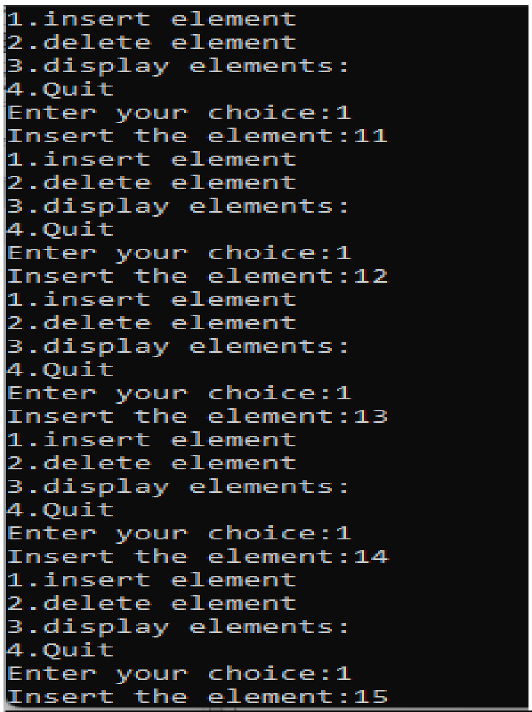
```
[null,null,null,null,-3,null,0,-2]
```

Lab Program 3:

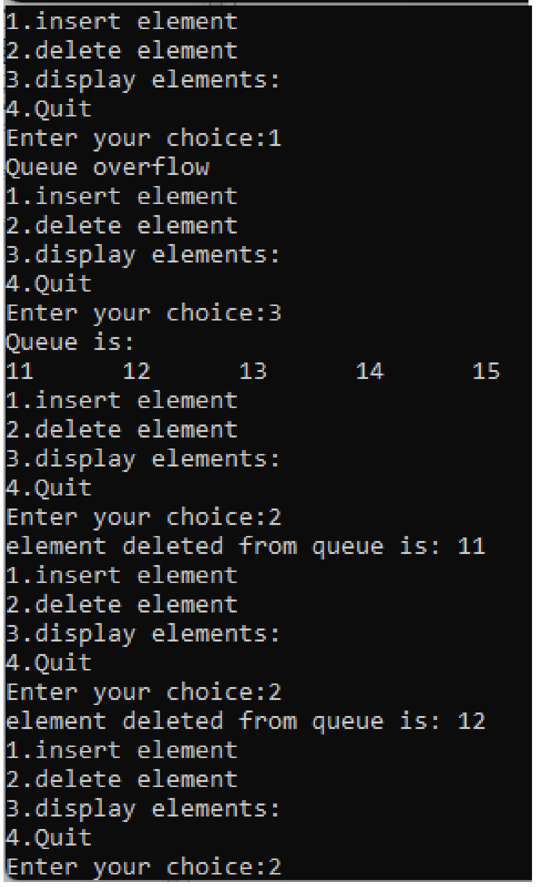
3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include<stdio.h>
#define MAX 5
void insert();
void delete();
void display();
int queue_array[MAX];
int rear=-1;
int front=-1;
main()
{
    int choice;
    while(1)
    {
        printf("1.insert element \n");
        printf("2.delete element \n");
        printf("3.display elements:\n");
        printf("4.Quit \n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            insert();
            break;
        case 2:
            delete();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(1);
        default:
            printf("Wrong choice \n");
        }
    }
}
void insert()
{
    int add_item;
    if(rear==MAX-1)
    {
        printf("Queue overflow \n");
    }
    else
    {
        if(front==-1)
            front=0;
            printf("Insert the element:");
        scanf("%d",&add_item);
        rear=rear+1;
        queue_array[rear]=add_item;
    }
}
void delete()
{
    if(front==-1 || front>rear)
    {
        printf("Queue underflow \n");
        return ;
    }
    else
    {
        printf("element deleted from queue is: %d \n",queue_array[front]);
        front=front+1;
    }
}
void display()
{
    int i;
    if(front==-1)
    {
        printf("Queue is empty \n");
    }
    else
    {
        printf("Queue is:\n");
        for(i=front; i<=rear; i++)
            printf("%d",queue_array[i]);
        printf("\n");
    }
}
```

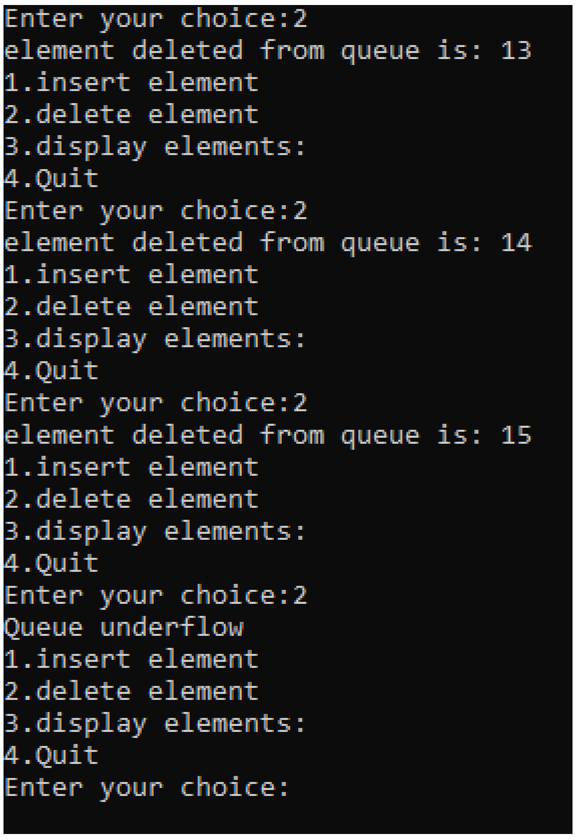
OUTPUT:



```
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:1
Insert the element:11
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:1
Insert the element:12
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:1
Insert the element:13
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:1
Insert the element:14
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:1
Insert the element:15
```



```
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:1
Queue overflow
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:3
Queue is:
11      12      13      14      15
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:2
element deleted from queue is: 11
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:2
element deleted from queue is: 12
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:2
```

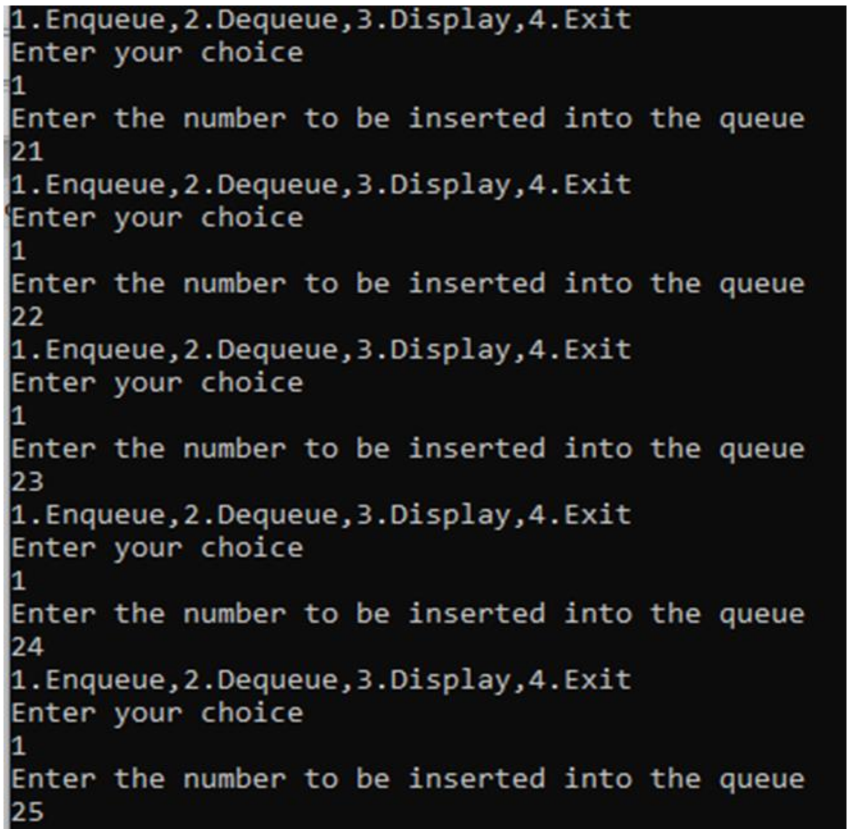


```
Enter your choice:2
element deleted from queue is: 13
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:2
element deleted from queue is: 14
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:2
element deleted from queue is: 15
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:2
Queue underflow
1.insert element
2.delete element
3.display elements:
4.Quit
Enter your choice:
```

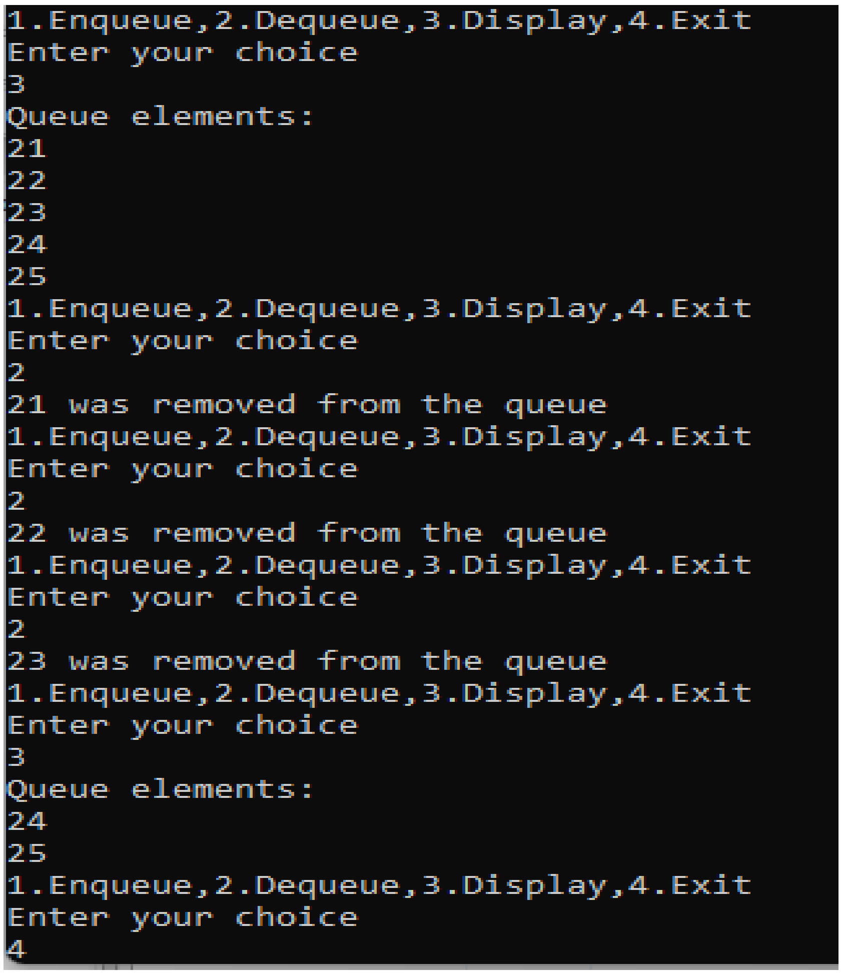
3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete &amp; Display The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include<stdio.h>
#include<stdlib.h>
#define size 5
int Q[size];
int rear=-1;
int front=-1;
int IsFull()
{
   if(front==(rear+1)%size)
   {
       return 0;
   }
   else
   {
       return -1;
   }
}
int IsEmpty()
{
    if(front==-1 && rear==-1)
    {
        return 0;
    }
    else
    {
        return -1;
    }
}
void Enqueue(int x)
{
   int item;
   if(IsFull()==0)
   {
       printf("Queue Overflow \n");
       return;
   }
   else
   {
      if(IsEmpty()==0)
      {
         front=0;
         rear=0;
      }
      else
      {
        rear=(rear+1)%size;
      }
      Q[rear]=x;
   }
}
int Dequeue()
{
    int x;
    if(IsEmpty()==0)
    {
        printf("Queue underflow \n");
    }
    else
    {
        if(front==rear)
        {
            x=Q[front];
            front=-1;
            rear=-1;
        }
        else
        {
            x=Q[front];
            front=(front+1)%size;
        }
        return x;
    }
}
void Display()
{
    int i;
    if(IsEmpty()==0)
    {
       printf("Queue is empty \n");
    }
    else
    {
        printf("Queue elements: \n");
        for(i=front; i!=rear; i=(i+1)%size)
        {
            printf("%d \n",Q[i]);
        }
        printf("%d \n",Q[i]);
    }
}
void main()
{
    int choice,x,b;
    while(1)
    {
        printf("1.Enqueue,2.Dequeue,3.Display,4.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
           case 1:printf("Enter the number to be inserted into the queue\n");
                  scanf("%d",&x);
                  Enqueue(x);
                  break;
           case 2:b=Dequeue();
                  printf("%d was removed from the queue\n",b);
                  break;
           case 3:Display();
                  break;
           case 4:exit(1);
           default:printf("Invalid input\n");

        }
    }
}
```

OUTPUT:



```
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
1
Enter the number to be inserted into the queue
21
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
1
Enter the number to be inserted into the queue
22
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
1
Enter the number to be inserted into the queue
23
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
1
Enter the number to be inserted into the queue
24
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
1
Enter the number to be inserted into the queue
25
```



```
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
3
Queue elements:
21
22
23
24
25
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
2
21 was removed from the queue
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
2
22 was removed from the queue
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
2
23 was removed from the queue
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
3
Queue elements:
24
25
1.Enqueue,2.Dequeue,3.Display,4.Exit
Enter your choice
4
```

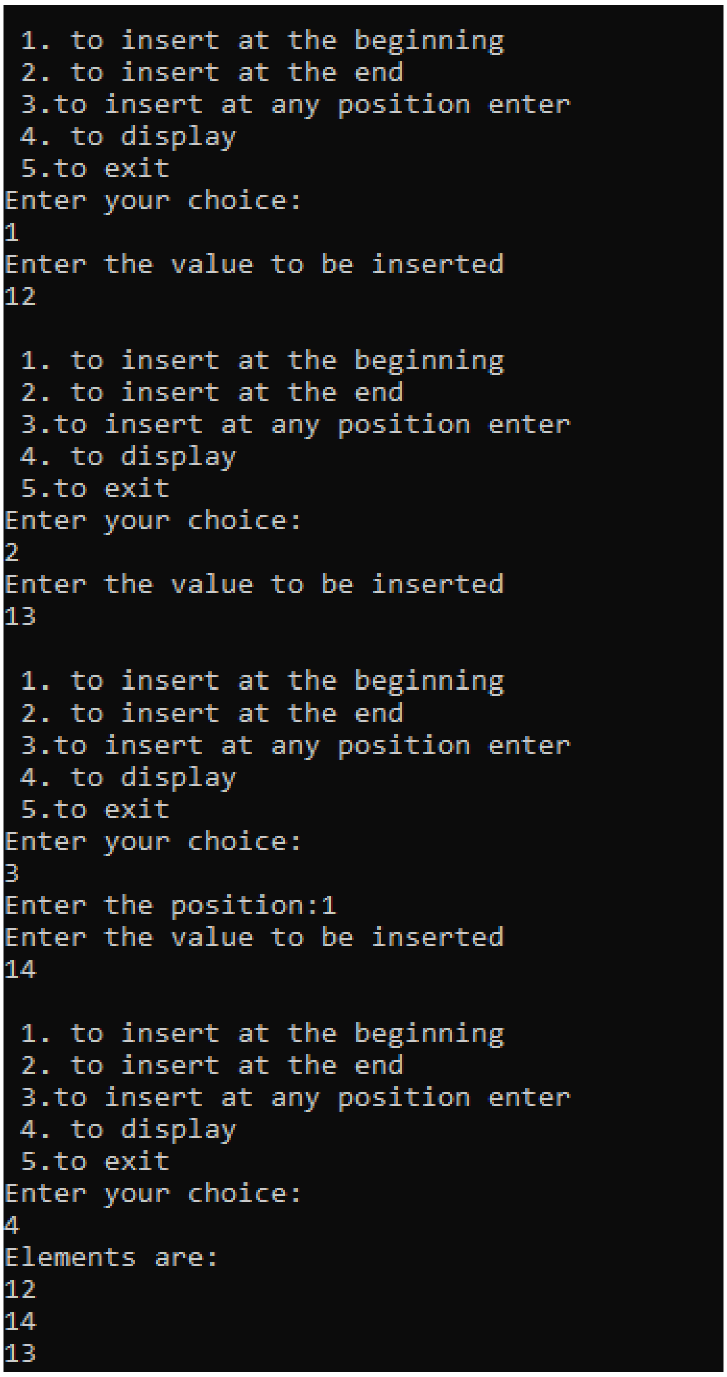
Lab Program 4

4 a)
WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
void display();
void insert_begin();
void insert_end();
void insert_pos();
struct node *head=NULL;
void display()
{
   printf("Elements are: \n");
   struct node *ptr;
   if(head==NULL)
   {
      printf("List is empty \n");
      return;
   }
   else
   {
      ptr=head;
      while(ptr!=NULL)
      {
         printf("%d \n", ptr->data);
         ptr=ptr->next;
      }
   }
}
void insert_begin()
{
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   printf("Enter the value to be inserted \n");
   scanf("%d", &temp->data);
   temp->next=NULL;
   if(head=NULL)
   {
      head=temp;
   }
```

```
    else
    {
        temp->next=head;
        head=temp;
    }
}
void insert_end()
{
    struct node *temp,*ptr;
    temp=(struct node*)malloc(sizeof(struct node));
    printf("Enter the value to be inserted \n");
    scanf("%d", &temp->data);
    temp->next=NULL;
    if(head==NULL)
    {
        head=temp;
    }
    else
    {
        ptr=head;
        while(ptr->next!=NULL)
        {
            ptr=ptr->next;
        }
        ptr->next=temp;
    }
}
void insert_pos()
{
    int pos,i;
    struct node *ptr, *temp;
    printf("Enter the position:");
    scanf("%d", &pos);
    temp=(struct node*)malloc(sizeof(struct node));
    printf("Enter the value to be inserted \n");
    scanf("%d", &temp->data);
    temp->next=NULL;
    if(pos==0)
    {
        temp->next=head;
        head=temp;
    }
    else
    {
        for(i=0,ptr=head; i<pos-1; i++)
        {
            ptr=ptr->next;
        }
        temp->next=ptr->next;
        ptr->next=temp;
    }
}
void main()
{
  int choice;
  while(1)
  {
    printf("\n 1. to insert at the beginning\n 2. to insert at the end\n 3.to insert at any position
enter\n 4. to display\n 5.to exit\n");
  printf("Enter your choice:\n");
  scanf("%d",&choice);
  switch(choice)
  {
    case 1:insert_begin();
           break;
    case 2:insert_end();
           break;
    case 3:insert_pos();
           break;
    case 4:display();
           break;
    case 5:exit(0);
           break;
    deafult:printf("Enter the correct choice\n");
            break;
  }
  }
}
```

OUTPUT:

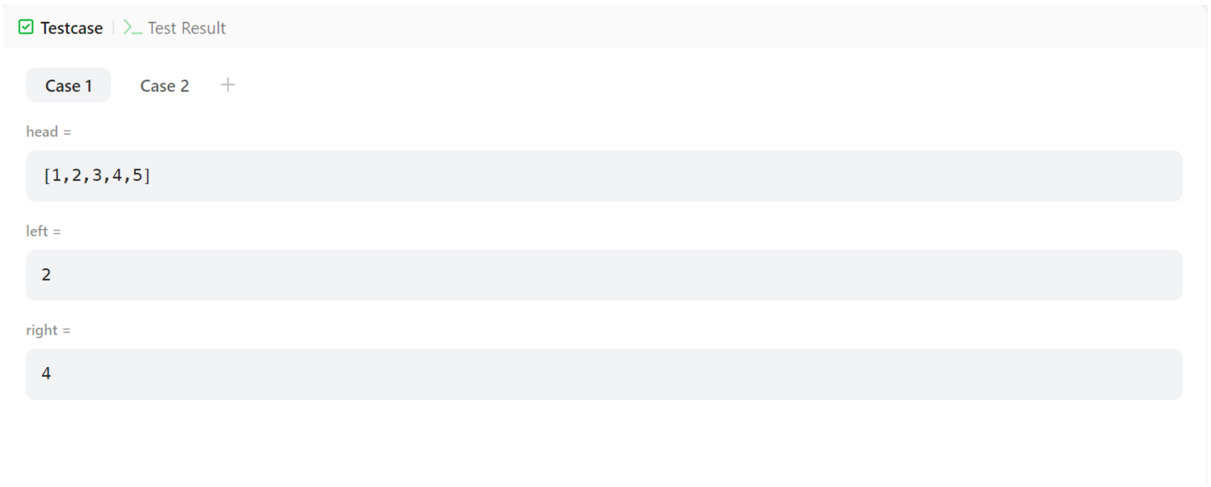


```
 1. to insert at the beginning
 2. to insert at the end
 3.to insert at any position enter
 4. to display
 5.to exit
Enter your choice:
1
Enter the value to be inserted
12

 1. to insert at the beginning
 2. to insert at the end
 3.to insert at any position enter
 4. to display
 5.to exit
Enter your choice:
2
Enter the value to be inserted
13

 1. to insert at the beginning
 2. to insert at the end
 3.to insert at any position enter
 4. to display
 5.to exit
Enter your choice:
3
Enter the position:1
Enter the value to be inserted
14

 1. to insert at the beginning
 2. to insert at the end
 3.to insert at any position enter
 4. to display
 5.to exit
Enter your choice:
4
Elements are:
12
14
13
```

4 b)Program - Leetcode platform-Reverse Linked List II
Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    if (head == NULL || left == right) {
        return head;
    }
    struct ListNode* p = malloc(sizeof(struct ListNode));
    p->next = head;
    struct ListNode *ptr,*ptr1 ;
    ptr1=p;
    ptr= head;

    for (int i = 1; i < left; i++) {
        ptr1= ptr;
        ptr = ptr->next;
    }

    struct ListNode *first = ptr;
    struct ListNode *last = ptr1;
    struct ListNode *next = NULL;

    for (int i = left; i <=right; i++) {
        struct ListNode *temp = ptr->next;
        ptr->next = next;
        next = ptr;
        ptr = temp;
    }

    last->next = next;
    first->next = ptr;
    return p->next;
}
```

Output:
Test case1:


Testcase | Test Result

Case 1 Case 2

head =

```
[1,2,3,4,5]
```

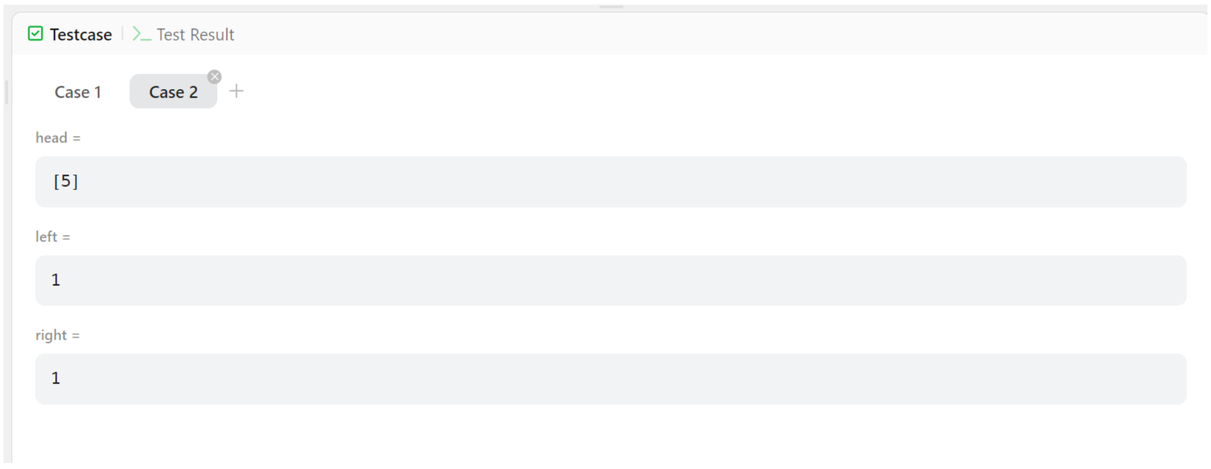
left =

```
2
```

right =

```
4
```

Test case2:


Testcase | Test Result

Case 1 Case 2

head =

```
[5]
```

left =

```
1
```

right =

```
1
```

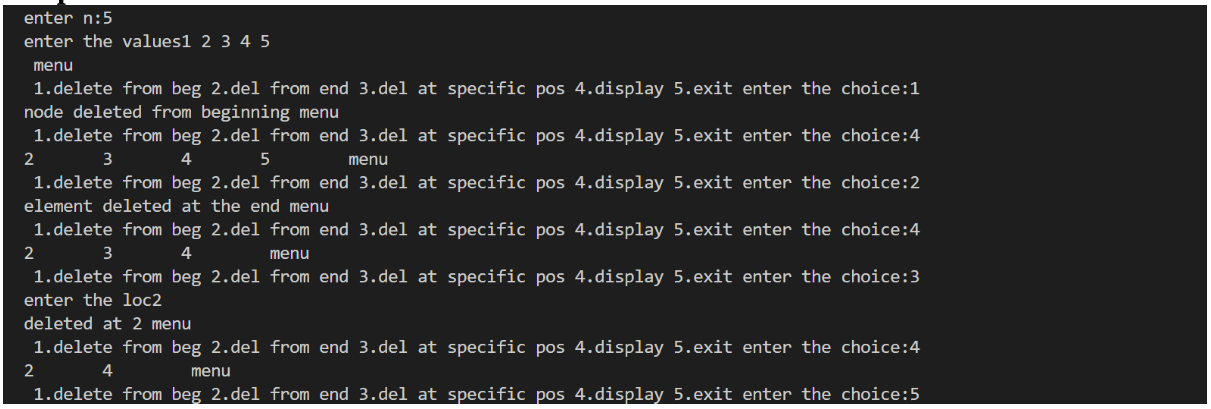
Lab Program 5

5a)
WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Deletion of first element, specified element and last element in the list.
c) Display the contents of the linked list.

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

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

void delbeg (struct node *a)
{
    struct node *ptr;
    if(head==NULL)
    {
        printf("list is empty");
    }
    else
    {
        ptr=head;
        head=ptr->next;
        free(ptr);
        printf("node deleted from beginning");

    }
}
void delend (struct node *a)
{
    struct node *ptr,*ptr1;
    if(head==NULL)
    {
        printf("list is empty");
    }
    else if(head->next==NULL)
    {
        free(head);
        head=NULL;
        printf("only one node is present and its deleted");
    }
    else
    {
        ptr=head;
        while(ptr->next!=NULL)
        {
            ptr1=ptr;
            ptr=ptr->next;
        }
        ptr1->next=NULL;
        free(ptr);
        printf("element deleted at the end");

    }
}
void delpos (struct node *a,int pos)
{
    if(head==NULL)
    {
        printf("list is empty");
    }
    else{
        int loc=pos;
        struct node *ptr,*ptr1;
        ptr=head;
        if (loc == 1)
        {
            head = ptr->next;
            free(ptr);
            printf("Deleted at position %d\n", loc);
            return;
        }
        for(int i=0;i<loc-1;i++)
        {
            ptr1=ptr;
            ptr=ptr->next;
            if(ptr==NULL)
            {
                printf("there are less than %d elements",loc);
                return;

            }
        }
        ptr1->next=ptr->next;
        free(ptr);
        printf("deleted at %d",loc);



    }
}
void display(struct node *s)
{
    while(s!=NULL)
    {
        printf("%d\t",s->data);
        s=s->next;
    }
}
void create(int a[],int n)
{
    struct node *last,*t;
    head=(struct node *) malloc (sizeof(struct node));
    head->data=a[0];
    head->next=NULL;
    last=head;
    for(int i=1;i<n;i++)
    {
        t=(struct node *) malloc (sizeof(struct node));
        t->data=a[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }

}
void main()
{
    int a[10],n;
    printf("enter n:");
    scanf("%d",&n);
    printf("enter the values");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    create(a,n);
    int c,l;
    while(1)
    {
        printf(" menu \n 1.delete from beg 2.del from end 3.del at specific pos 4.display 5.exit
");
        printf("enter the choice:");
        scanf("%d",&c);
        switch(c)
        {
            case 1:delbeg(head);
                    break;
            case 2:delend(head);
                    break;
            case 3:printf("enter the loc");
            scanf("%d",&l);
                    delpos(head,l);
                    break;
            case 4:display(head);
```

```
                break;
            case 5:exit(0);
            default:printf("invalid input");
                break;

        }
    }
}
```

Output:



```
enter n:5
enter the values1 2 3 4 5
 menu
 1.delete from beg 2.del from end 3.del at specific pos 4.display 5.exit enter the choice:1
node deleted from beginning menu
 1.delete from beg 2.del from end 3.del at specific pos 4.display 5.exit enter the choice:4
2       3       4       5        menu
 1.delete from beg 2.del from end 3.del at specific pos 4.display 5.exit enter the choice:2
element deleted at the end menu
 1.delete from beg 2.del from end 3.del at specific pos 4.display 5.exit enter the choice:4
2       3       4        menu
 1.delete from beg 2.del from end 3.del at specific pos 4.display 5.exit enter the choice:3
enter the loc2
deleted at 2 menu
 1.delete from beg 2.del from end 3.del at specific pos 4.display 5.exit enter the choice:4
2       4        menu
 1.delete from beg 2.del from end 3.del at specific pos 4.display 5.exit enter the choice:5
```

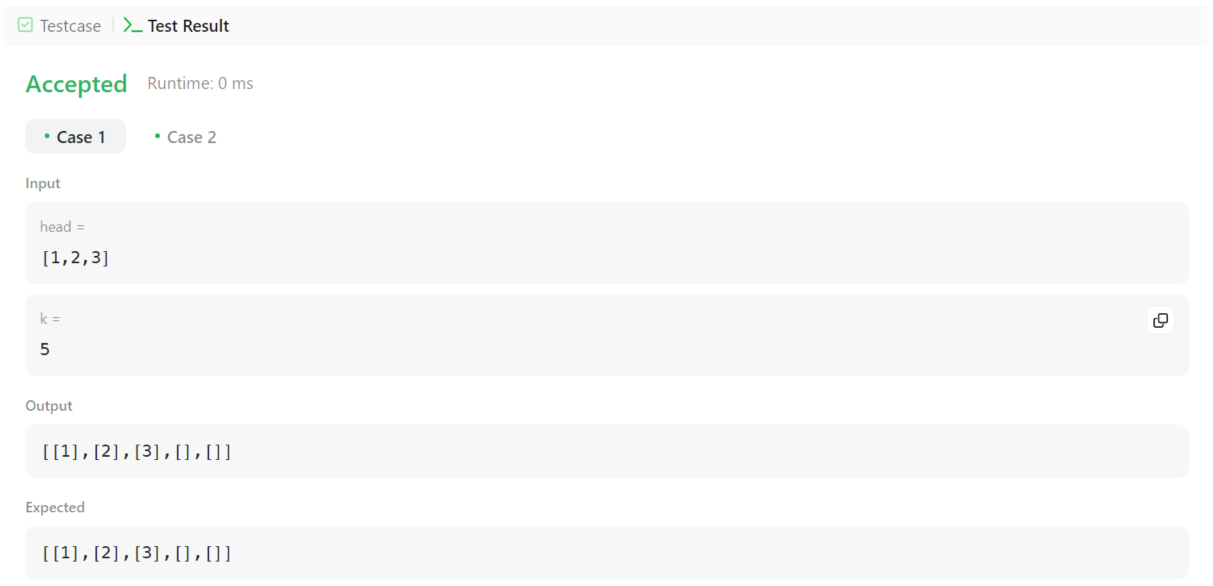
5b) Program - Leetcode platform-Split Linked List in Parts

Given the head of a singly linked list and an integer k, split the linked list into k consecutive linked list parts.
The length of each part should be as equal as possible: no two parts should have a size differing by more than one. This may lead to some parts being null.
The parts should be in the order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal to parts occurring later.
Return an array of the k 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) {

     *returnSize = k;
    struct ListNode** arr = (struct ListNode*)malloc(k*sizeof(struct ListNode));
        for (int i=0; i<k; i++){
            arr[i] = (struct ListNode*)malloc(sizeof(struct ListNode));
        }
    int i=0,size=0,c;
    struct ListNode* curr=head;
    struct ListNode* temp;
    while (curr){
        size++;
        curr = curr->next;
    }
    while (k && size){
        c = size/k;
        if (size % k != 0){
            c++;
        }
        curr = head;
        temp = head;
        for (int j=1; j<c; j++){
            temp = temp->next;
        }
        head = temp->next;
        temp->next = NULL;
        arr[i] = curr;
        i++;
        k--;
        size -= c;
    }
    if (k){
        for (int j=0; j<k; j++){
            arr[i] = NULL;
            i++;
        }
    }

    return arr;

}
```

Output:

Test case1:



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

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

k =

```
5
```

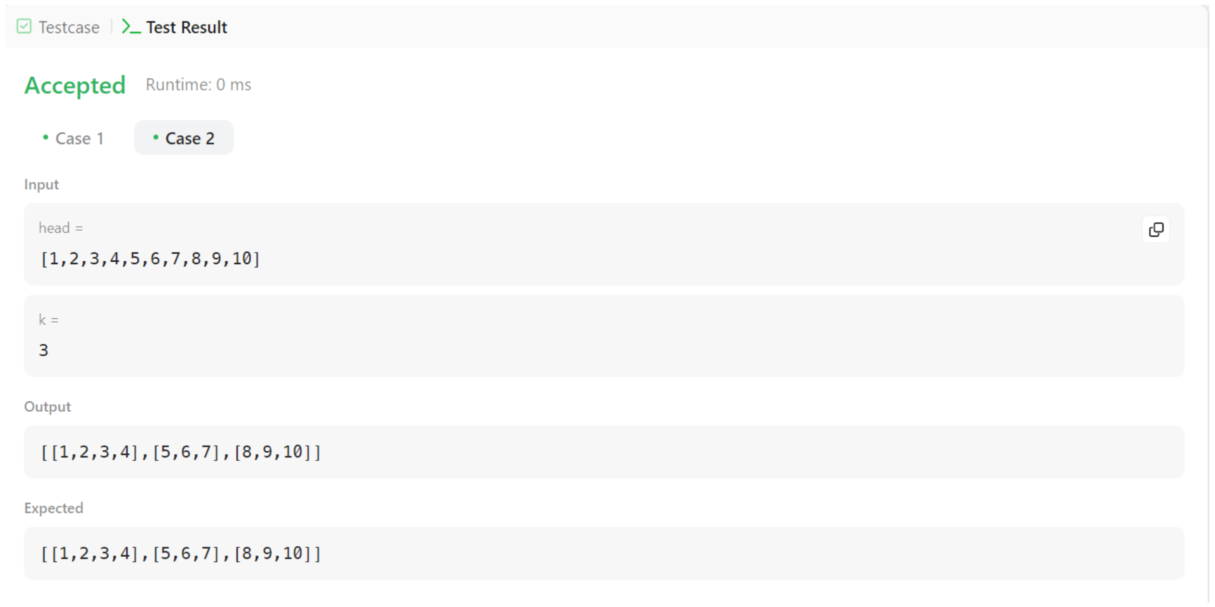
Output

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

Expected

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

Test case2:



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

```
[1,2,3,4,5,6,7,8,9,10]
```

k =

```
3
```

Output

```
[[1,2,3,4],[5,6,7],[8,9,10]]
```

Expected

```
[[1,2,3,4],[5,6,7],[8,9,10]]
```

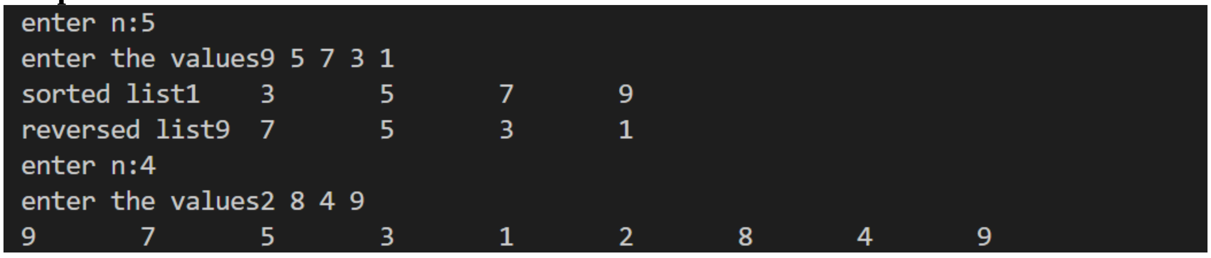
Lab Program 6

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

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
}*first=NULL,*second=NULL;
void display(struct node *s)
{
   while(s!=NULL)
   {
      printf("%d\t",s->data);
      s=s->next;
   }
}
void create1(int a[],int n)
{
   struct node *last,*t;
   first=(struct node *) malloc (sizeof(struct node));
   first->data=a[0];
   first->next=NULL;
   last=first;
   for(int i=1;i<n;i++)
   {
      t=(struct node *) malloc (sizeof(struct node));
      t->data=a[i];
      t->next=NULL;
      last->next=t;
      last=t;
   }

}
void create2(int a[],int n)
{
   struct node *last,*t;
   second=(struct node *) malloc (sizeof(struct node));
   second->data=a[0];
   second->next=NULL;
   last=second;
   for(int i=1;i<n;i++)
   {
      t=(struct node *) malloc (sizeof(struct node));
      t->data=a[i];
      t->next=NULL;
      last->next=t;
```

```
        last=t;
    }

}
void reverse(struct node *p)
{
    struct node *q,*r;
    p=first;
    q=NULL;
    r=NULL;
    while(p!=NULL)
    {
        r=q;
        q=p;
        p=p->next;
        q->next=r;

    }
    first=q;
}
struct node* concatat(struct node *p,struct node *q)
{
    struct node *r;
    if(p==NULL)
    {
        p=q;
        return p;
    }
    if(q==NULL)
    {
        return q;
    }
    r=p;
    while(r->next!=NULL)
        r=r->next;
    r->next=q;
    return p;

}
void sort(struct node *p)
{
    struct node *i,*j;
    int temp;
    for(i=p;i->next!=NULL;i=i->next)
    {
        for(j=i->next;j!=NULL;j=j->next)
        {
            if(i->data>j->data)
            {
                temp=i->data;
                i->data=j->data;
                j->data=temp;
            }
          }}
}
void main()
{
    int a[10],b[10],n1,n2;
    printf("enter n:");
    scanf("%d",&n1);
    printf("enter the values");
    for(int i=0;i<n1;i++)
    {
        scanf("%d",&a[i]);
    }
    struct node *s;
    s=(struct node *) malloc (sizeof(struct node));
    create1(a,n1);
    sort(first);
    printf("sorted list");
    display(first);
    reverse(first);
    printf("\nreversed list");
    display(first);
    printf("\nenter n:");
    scanf("%d",&n2);
    printf("enter the values");
    for(int i=0;i<n2;i++)
    {
        scanf("%d",&b[i]);
    }
    create2(b,n2);
    // display(second);
    s=concatat(first,second);
    display(s);
}
```

Output:

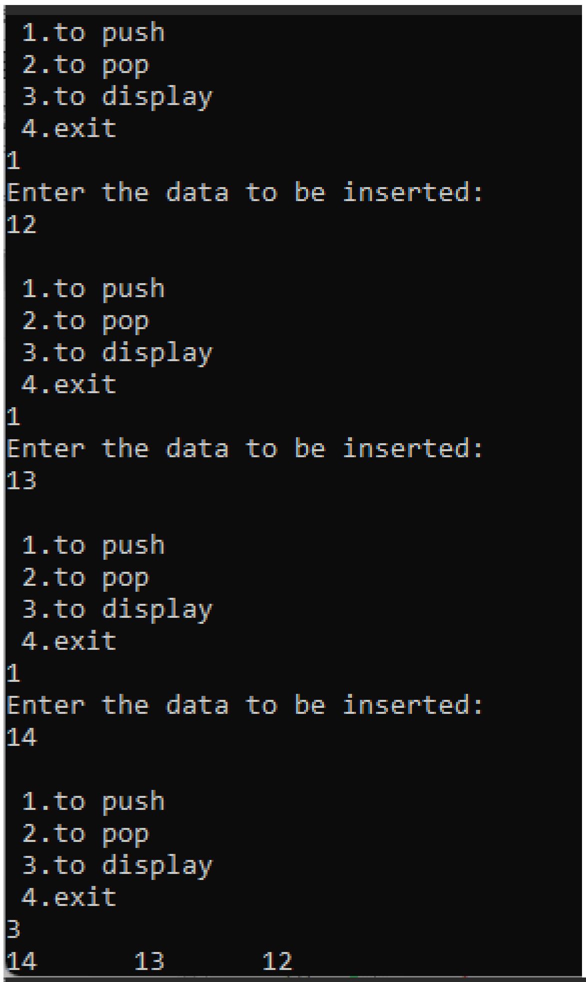


```
enter n:5
enter the values9 5 7 3 1
sorted list1    3       5       7       9
reversed list9  7       5       3       1
enter n:4
enter the values2 8 4 9
9       7       5       3       1       2       8       4       9
```

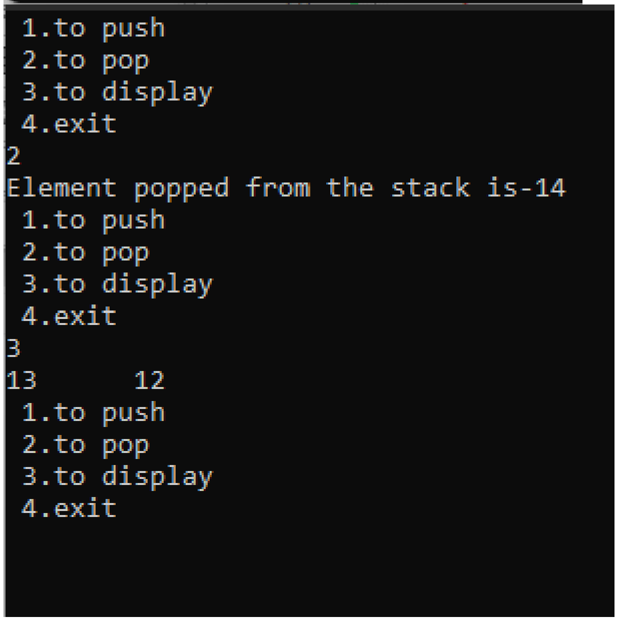
6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

//Stack
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head=NULL;
void push()
{
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   printf("Enter the data to be inserted: \n");
   scanf("%d",&temp->data);
   temp->next=NULL;
   if(head==NULL)
   {
      head=temp;
      return;
   }
   else
   {
      temp->next=head;
      head=temp;
   }
}
void pop()
{
   struct node *ptr;
   ptr=head;
   if(head==NULL)
   {
      printf("Stack is empty item can't be popped \n");
      return;
   }
   else
   {
      ptr=head;
      head=ptr->next;
      printf("Element popped from the stack is-%d",ptr->data);
      free(ptr);
   }
}
void display()
{
   if(head==NULL)
   {
        printf("Stack is empty \n");
        return;
    }
    else
    {
        struct node *ptr;
        ptr=head;
        while(ptr!=NULL)
        {
            printf("%d \t",ptr->data);
            ptr=ptr->next;
        }
    }
}
void main()
{
    int choice;
    while(1)
    {
        printf("\n 1.to push \n 2.to pop \n 3.to display \n 4.exit \n");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        }
    }
}
```

OUTPUT:



```
 1.to push
 2.to pop
 3.to display
 4.exit
1
Enter the data to be inserted:
12

 1.to push
 2.to pop
 3.to display
 4.exit
1
Enter the data to be inserted:
13

 1.to push
 2.to pop
 3.to display
 4.exit
1
Enter the data to be inserted:
14

 1.to push
 2.to pop
 3.to display
 4.exit
3
14      13      12
```



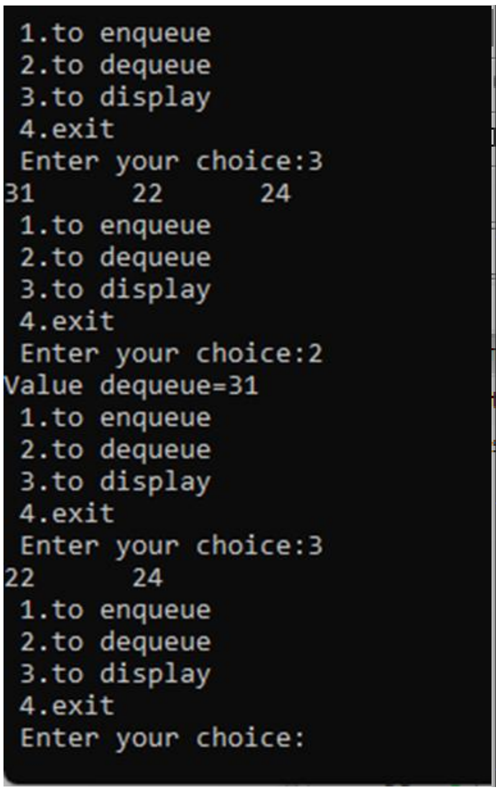
```
 1.to push
 2.to pop
 3.to display
 4.exit
2
Element popped from the stack is-14
 1.to push
 2.to pop
 3.to display
 4.exit
3
13      12
 1.to push
 2.to pop
 3.to display
 4.exit
```

//Queue

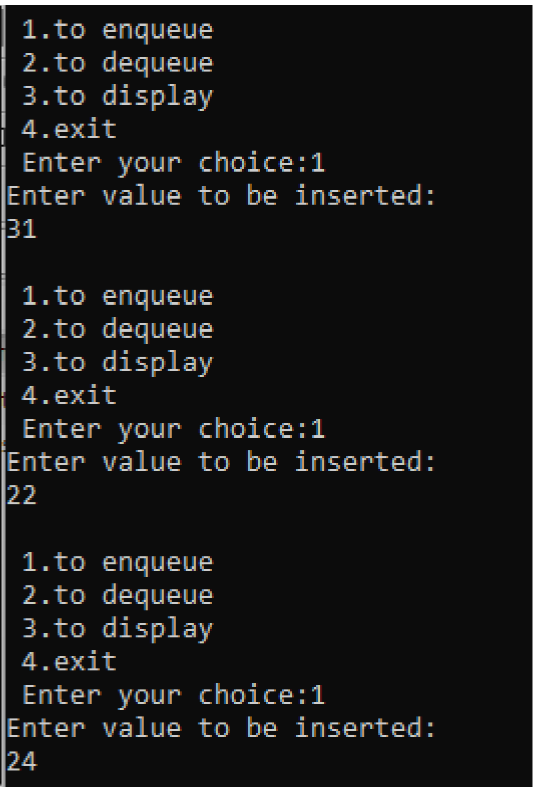
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head=NULL;

void enqueue()
{
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   printf("Enter value to be inserted: \n");
   scanf("%d",&temp->data);
   temp->next=NULL;
   if(head==NULL)
   {
      head=temp;
      return;
   }
   else
   {
      struct node *ptr;
      ptr=head;
      while(ptr->next!=NULL)
      {
         ptr=ptr->next;
      }
      ptr->next=temp;
   }
}

void dequeue()
{
   struct node *ptr;
   if(head==NULL)
   {
      printf("Queue is empty \n");
      return;
   }
```

```
    else
    {
        ptr=head;
        head=ptr->next;
        printf("Value dequeue=%d",ptr->data);
        free(ptr);
    }
}

void display()
{
    if(head==NULL)
    {
        printf("Queue is empty \n");
        return;
    }
    else
    {
        struct node *ptr;
        ptr=head;
        while(ptr!=NULL)
        {
            printf("%d \t",ptr->data);
            ptr=ptr->next;
        }
    }
}

void main()
{
    int choice;
    while(1)
    {
        printf("\n 1.to enqueue \n 2.to dequeue\n 3.to display \n 4.exit\n ");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        }
    }
}
```

OUTPUT:



```
 1.to enqueue
 2.to dequeue
 3.to display
 4.exit
 Enter your choice:3
31      22      24
 1.to enqueue
 2.to dequeue
 3.to display
 4.exit
 Enter your choice:2
Value dequeue=31
 1.to enqueue
 2.to dequeue
 3.to display
 4.exit
 Enter your choice:3
22      24
 1.to enqueue
 2.to dequeue
 3.to display
 4.exit
 Enter your choice:
```



```
 1.to enqueue
 2.to dequeue
 3.to display
 4.exit
 Enter your choice:1
Enter value to be inserted:
31

 1.to enqueue
 2.to dequeue
 3.to display
 4.exit
 Enter your choice:1
Enter value to be inserted:
22

 1.to enqueue
 2.to dequeue
 3.to display
 4.exit
 Enter your choice:1
Enter value to be inserted:
24
```

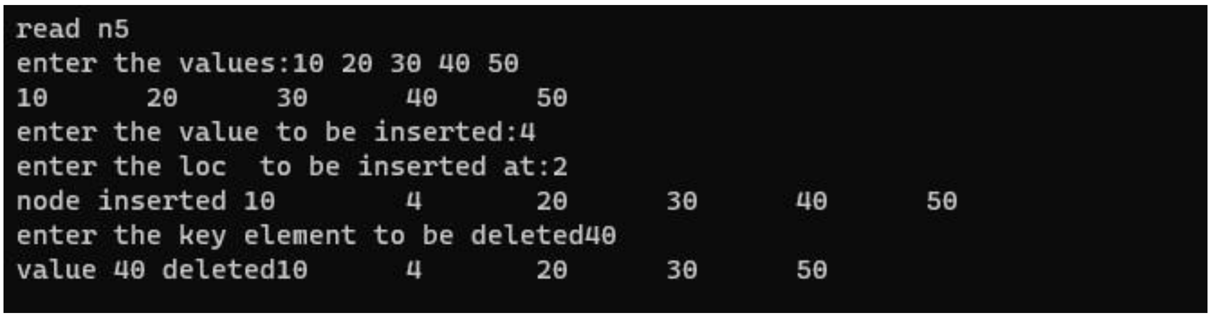
Lab Program 7
7 a)
WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *prev;
   struct node *next;
};
struct node *s1=NULL;

struct node *insert_begin(struct node *start)
{
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   printf("Enter the value to be inserted \n");
   scanf("%d",&temp->data);
   temp->next=NULL;
   temp->prev=NULL;
   if(start==NULL)
   {
      start=temp;
   }
   else
   {
      temp->next=start;
      start->prev=temp;
      start=temp;
   }
   return start;
}
struct node *insert_end(struct node *start)
{
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   printf("Enter the value to be inserted \n");
   scanf("%d",&temp->data);
   temp->next=NULL;
   temp->prev=NULL;
   if(start==NULL)
   {
      start=temp;
```

```
    }
    else
    {
      struct node *ptr;
      ptr=start;
      while(ptr->next!=NULL)
      {
          ptr=ptr->next;
      }
      ptr->next=temp;
      temp->prev=ptr;
    }
    return start;
}
struct node* delevalue(struct node *start, int val) {
    struct node *ptr = start;
    int value = val;

    while (ptr != NULL) {
        if (ptr->data == value) {
            if (ptr->prev != NULL) {
                ptr->prev->next = ptr->next;
            }
            if (ptr->next != NULL) {
                ptr->next->prev = ptr->prev;
            }
            if (ptr == start) {
                start = ptr->next;
            }
            free(ptr);
            printf("Value %d deleted\n", value);
            return start;
        }
        ptr = ptr->next;
    }
    printf("Value %d not found\n", value);
    return start;
}
void display(struct node *start) {
    struct node *ptr = start;
    if (start == NULL) {
        printf("\nList is empty\n");
    } else {
        while (ptr != NULL) {
            printf("%d\n", ptr->data);
            ptr = ptr->next;
        }
    }
}
int main()
```

```
{
   int choice;
   while(1)
   {
       printf("\n 1.to add in beginning \n 2.to add at end \n 3.to display \n 4.to delete \n 5.exit
\n");
       scanf("%d",&choice);
       switch(choice)
       {
       case 1:
           s1=insert_begin(s1);
           break;
       case 2:
           s1=insert_end(s1);
           break;
       case 3:
           display(s1);
           break;
       case 4:
           printf("Enter the value to delete: ");
           int val;
           scanf("%d", &val);
           s1 = delevalue(s1, val);
           break;
       case 5:
           exit(0);
       default:
           printf("Wrong choice");

       }
   }
}
```

Output:



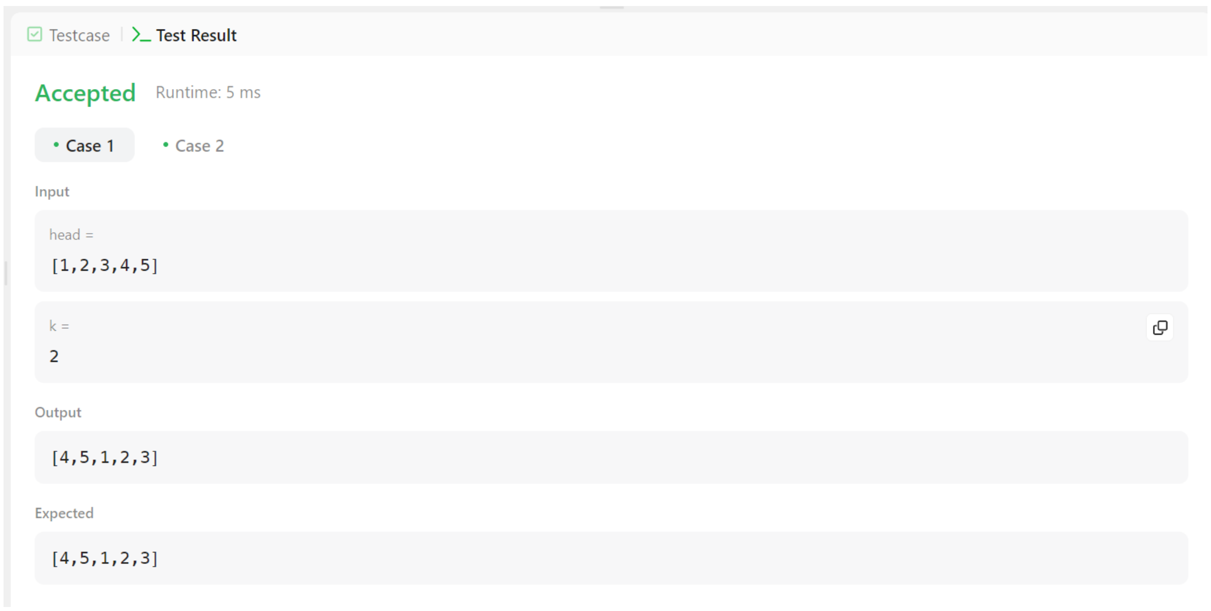
```
read n5
enter the values:10 20 30 40 50
10      20      30      40      50
enter the value to be inserted:4
enter the loc  to be inserted at:2
node inserted 10        4       20      30      40      50
enter the key element to be deleted40
value 40 deleted10      4       20      30      50
```

7b) Program - Leetcode platform-Rotate List

Given the head of a linked list, rotate the list to the right by k places.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* rotateRight(struct ListNode* head, int k) {

    struct ListNode* p,*q;
    if(head==NULL||head->next==NULL||k==0){
        return head;
    }
    p=head;
    int count=1;

    while(p->next!=NULL){
        p=p->next;
        count++;
    }
    k=k%count;
    if(k==0){
        return head;
    }
    p->next=head;
    p=head;

    for(int i=0;i<count-k-1;i++){
        p=p->next;
    }
    q=p->next;
    p->next=NULL;
    return q;

}
```

Output:
Testcase 1:


Testcase | Test Result

Accepted Runtime: 5 ms

• Case 1 • Case 2

Input

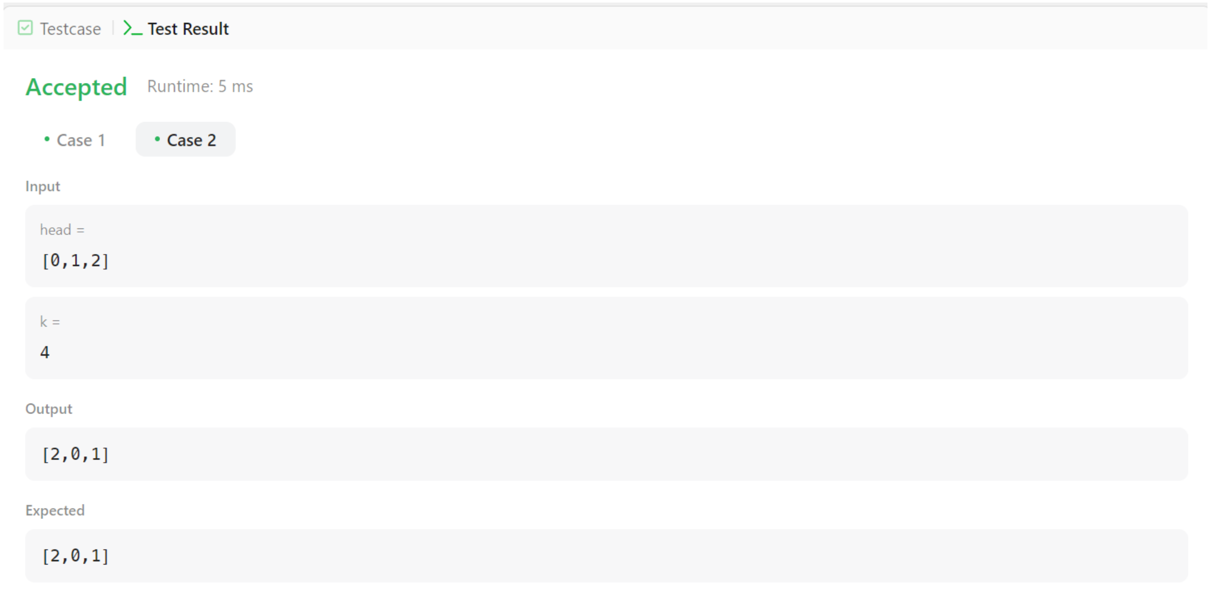
head =
```
[1,2,3,4,5]
```

k =
```
2
```

Output
```
[4,5,1,2,3]
```

Expected
```
[4,5,1,2,3]
```

Testcase 2:


Testcase | Test Result

Accepted Runtime: 5 ms

• Case 1 • Case 2

Input

head =
```
[0,1,2]
```

k =
```
4
```

Output
```
[2,0,1]
```

Expected
```
[2,0,1]
```

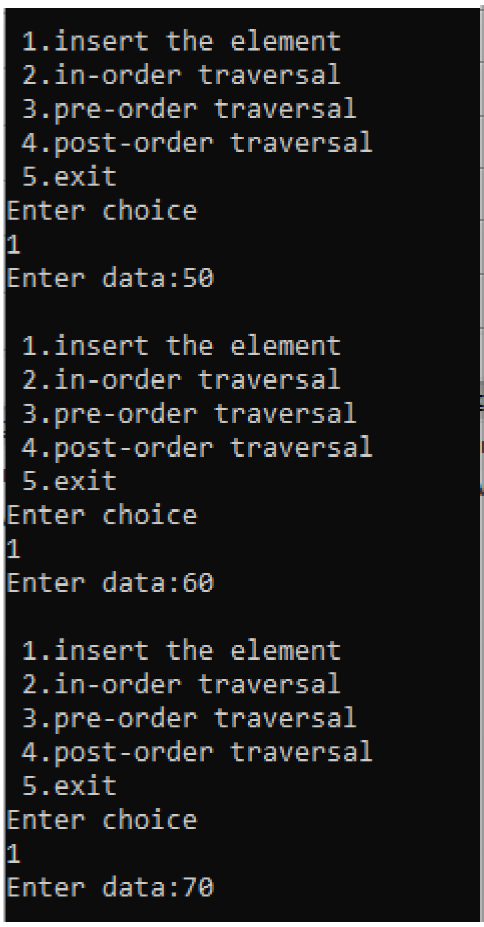
Lab Program 8

8 a)
Write a program
a) To construct a binary Search tree.
b) To traverse the tree using all the methods i.e., in-order, preorder and post order
c) To display the elements in the tree.

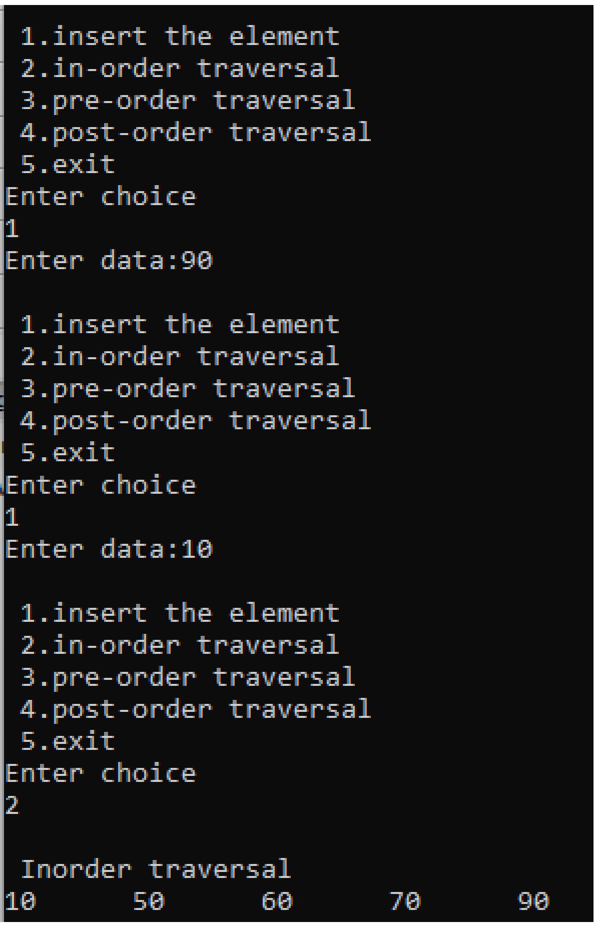
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int value;
   struct node *left;
   struct node *right;
}node;

struct node *create()
{
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   printf("Enter data:");
   scanf("%d",&temp->value);
   temp->left=temp->right=NULL;
   return temp;
}

void insert(struct node *root,struct node *temp)
{
   if(temp->value<root->value)
   {
     if(temp->left!=NULL)
      insert(root->left,temp);
   else
      root->left=temp;
   }
   if(temp->value>root->value)
   {
      if(root->right!=NULL)
         insert(root->right,temp);
      else
         root->right=temp;
   }
}
void inorder(struct node *root)
{
   if(root!=NULL)
   {
      inorder(root->left);
```

```
        printf("%d \t",root->value);
        inorder(root->right);
    }
}
void preorder(struct node *root)
{
    if(root!=NULL)
    {
        printf("%d \t",root->value);
        preorder(root->left);
        preorder(root->right);
    }
}
void postorder(struct node *root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d \t",root->value);
    }
}
int main()
{
    int choice;
    struct node *root=NULL;
    while(1)
    {
        printf("\n 1.insert the element \n 2.in-order traversal \n 3.pre-order traversal \n 4.post-
order traversal \n 5.exit");
        printf("\nEnter choice \n");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            if(root==NULL)
                root=create();
            else
                insert(root,create());
            break;
        case 2:
            printf("\n Inorder traversal \n");
            inorder(root);
            break;
        case 3:
            printf("\n Preorder traversal \n");
            preorder(root);
            break;
        case 4:
            printf("\n Postorder traversal \n");
            postorder(root);
            break;

        case 5:
            exit(0);
        default:
            printf("Invalid choice");
            break;
        }
    }
    return 0;
}
```

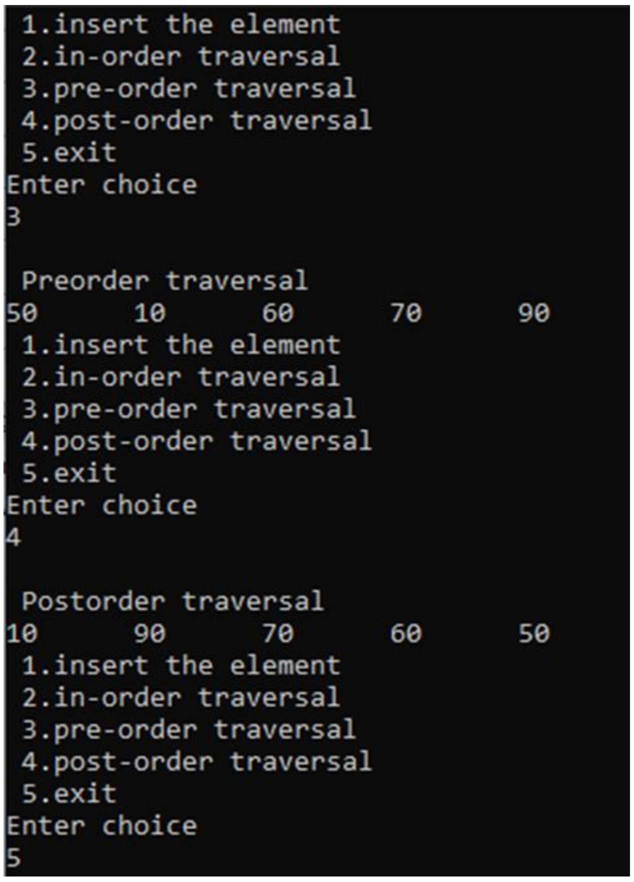
Output:



```
 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
1
Enter data:50

 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
1
Enter data:60

 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
1
Enter data:70
```



```
 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
1
Enter data:90

 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
1
Enter data:10

 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
2

 Inorder traversal
10      50      60      70      90
```



```
 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
3

 Preorder traversal
50      10      60      70      90
 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
4

 Postorder traversal
10      90      70      60      50
 1.insert the element
 2.in-order traversal
 3.pre-order traversal
 4.post-order traversal
 5.exit
Enter choice
5
```

8b) Program - Hackerrank platform

Swapping subtrees of a node means that if initially node has left subtree L and right subtree R, then after swapping, the left subtree will be R and the right subtree, L.

Complete the *swapNodes* function in the editor below. It should return a two-dimensional array where each element is an array of integers representing the node indices of an in-order traversal after a swap operation.

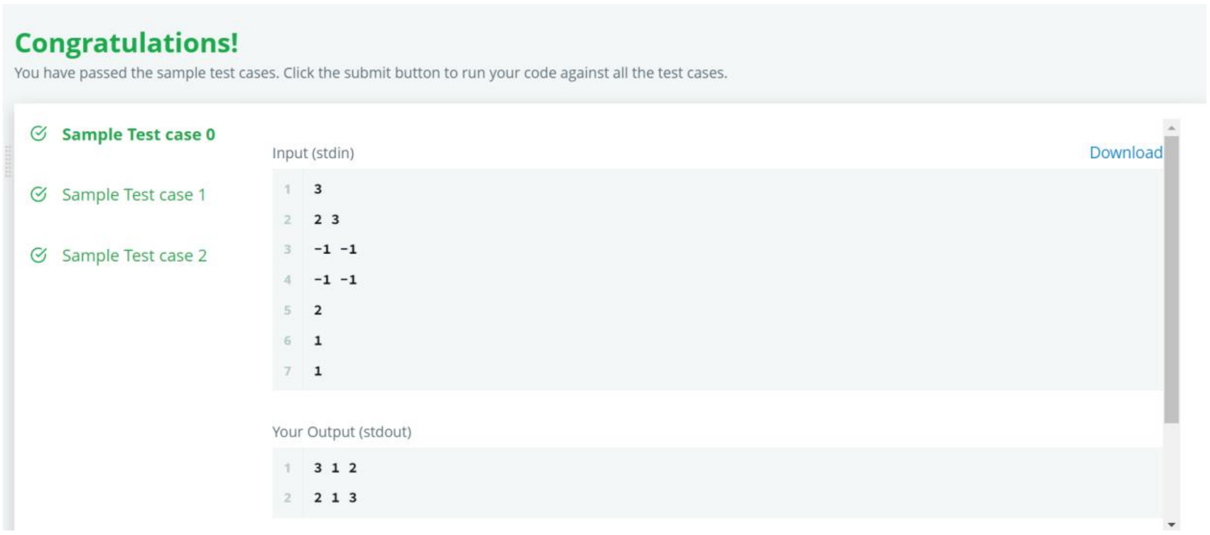
swapNodes has the following parameter(s):
- *indexes*: an array of integers representing index values of each , beginning with , the first element, as the root.
- *queries*: an array of integers, each representing a value.

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

struct node
{
   int id;
   int depth;
   struct node *left, *right;
};
```

```
void
inorder(struct node* tree)
{
    if(tree == NULL)
        return;

    inorder(tree->left);
    printf("%d ",tree->id);
    inorder((tree->right));

}


int
main(void)
{
    int no_of_nodes, i = 0;
    int l,r, max_depth,k;
    struct node* temp = NULL;
    scanf("%d",&no_of_nodes);
    struct node* tree = (struct node *) calloc(no_of_nodes , sizeof(struct node));
    tree[0].depth = 1;
    while(i <  no_of_nodes )
    {
        tree[i].id = i+1;
        scanf("%d %d",&l,&r);
        if(l ==  -1)
            tree[i].left = NULL;
        else
             {
                tree[i].left = &tree[l-1];
                tree[i].left->depth = tree[i].depth + 1;
                max_depth = tree[i].left->depth;
             }

         if(r ==  -1)
            tree[i].right = NULL;
        else
             {
                tree[i].right = &tree[r-1];
                tree[i].right->depth = tree[i].depth + 1;
                max_depth = tree[i].right->depth+2;
             }

    i++;
    }

    scanf("%d", &i);
    while(i--)
    {
        scanf("%d",&l);
        r = l;
        while(l <= max_depth)
        {
            for(k = 0;k < no_of_nodes; ++k)
            {
                if(tree[k].depth == l)
                {
                    temp = tree[k].left;
                    tree[k].left = tree[k].right;
                    tree[k].right = temp;
                }
            }
            l = l + r;
        }
        inorder(tree);
        printf("\n");
    }



    return 0;
}
```

Output:

Test case 1:



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0
Sample Test case 1
Sample Test case 2

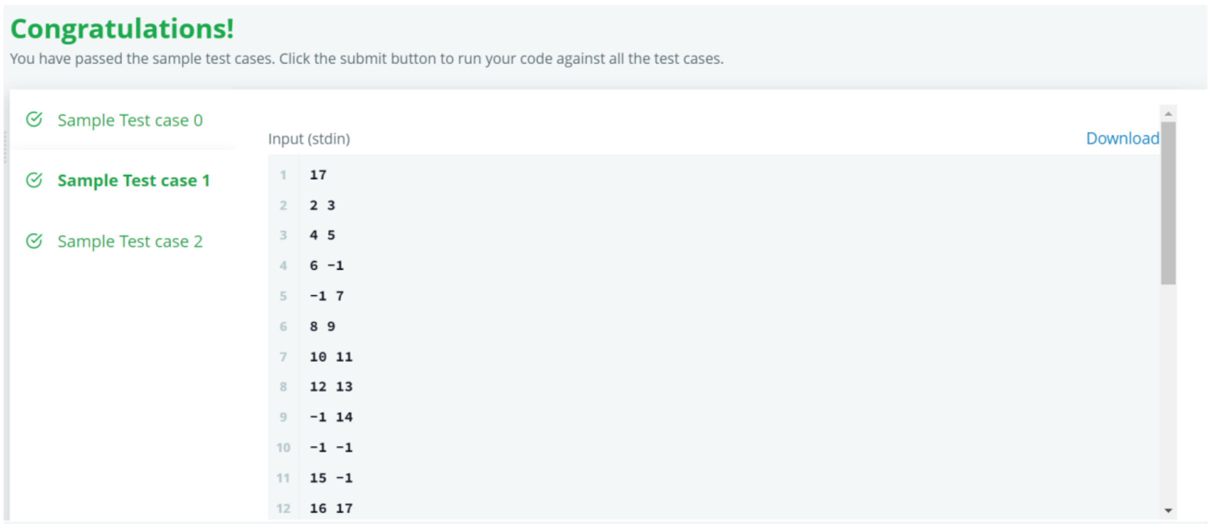
Input (stdin)

```
3
2 3
-1 -1
-1 -1
2
1
1
```

Your Output (stdout)

```
3 1 2
2 1 3
```

Test case 2:


Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

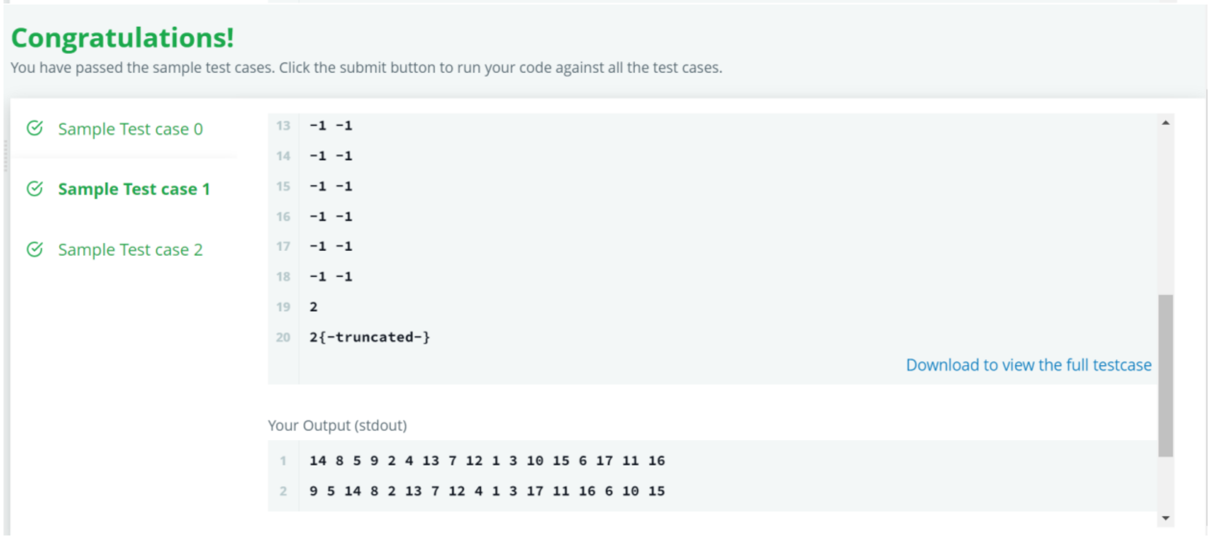
Sample Test case 0

Sample Test case 1

Sample Test case 2

Input (stdin) Download

```
17
2 3
4 5
6 -1
-1 7
8 9
10 11
12 13
-1 14
-1 -1
15 -1
16 17
```


Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Sample Test case 1

Sample Test case 2

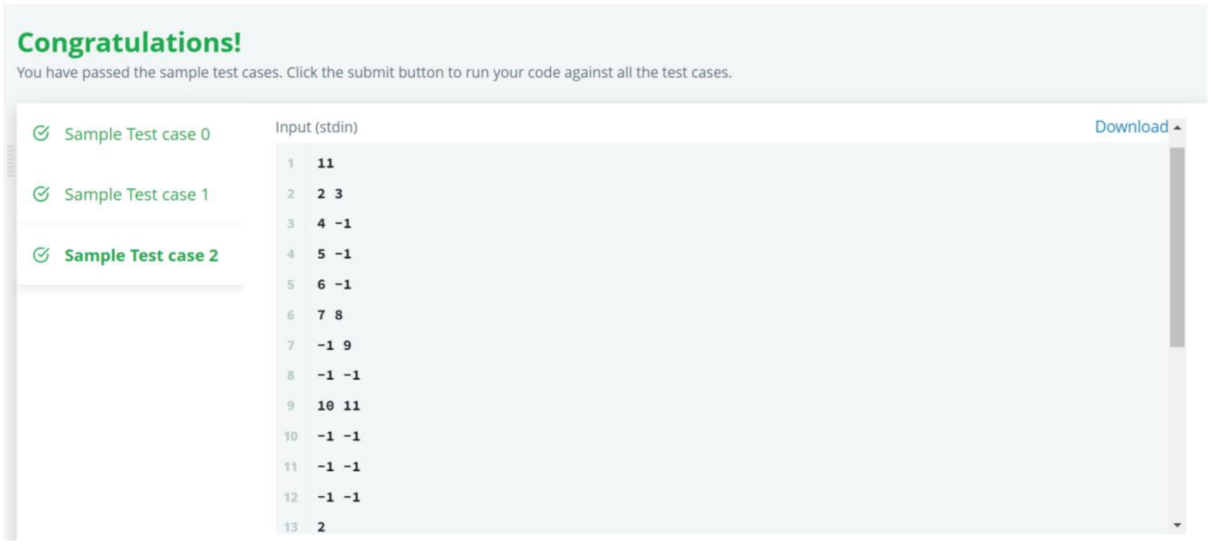
```
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
2
2{-truncated-}
```

Download to view the full testcase

Your Output (stdout)

```
14 8 5 9 2 4 13 7 12 1 3 10 15 6 17 11 16
9 5 14 8 2 13 7 12 4 1 3 17 11 16 6 10 15
```

Testcase 3:


Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

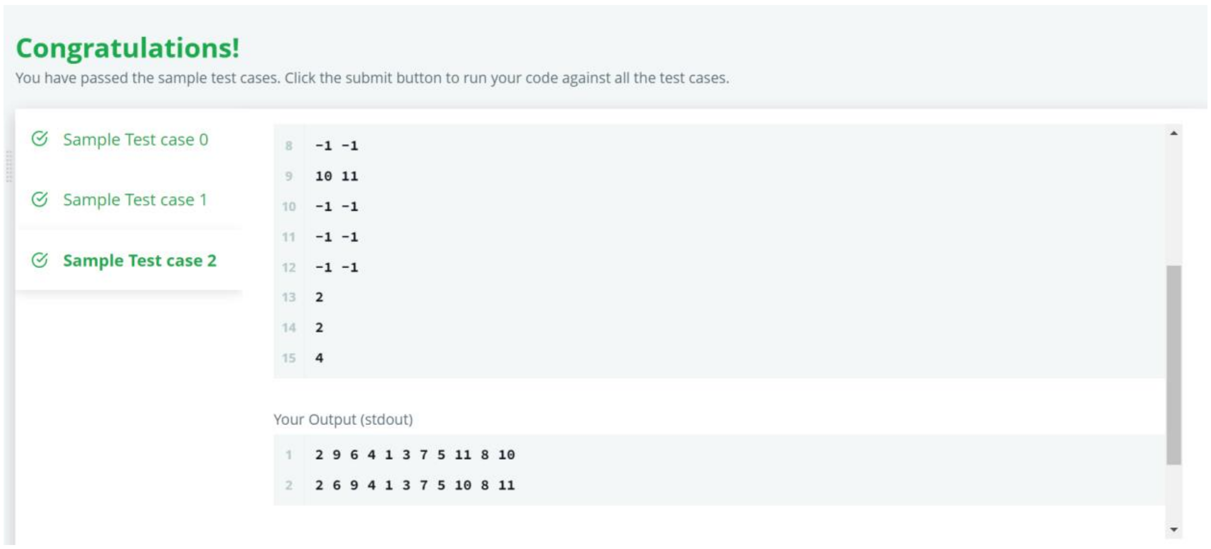
Sample Test case 0

Sample Test case 1

Sample Test case 2

Input (stdin) Download

```
11
2 3
4 -1
5 -1
6 -1
7 8
-1 9
-1 -1
10 11
-1 -1
-1 -1
-1 -1
2
```



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

```
-1 -1
10 11
-1 -1
-1 -1
-1 -1
2
2
4
```

Your Output (stdout)

```
2 9 6 4 1 3 7 5 11 8 10
2 6 9 4 1 3 7 5 10 8 11
```

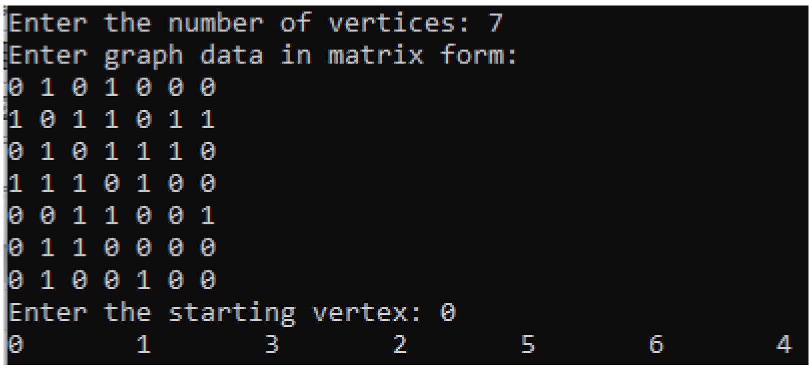
Lab Program 9

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

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

```
#include <stdio.h>

#define MAX_VERTICES 10

int n, i, j, visited[MAX_VERTICES], queue[MAX_VERTICES], front = 0, rear = 0;
int adj[MAX_VERTICES][MAX_VERTICES];

void bfs(int v) {
    visited[v] = 1;
    queue[rear++] = v;

    while (front < rear) {
        int current = queue[front++];
        printf("%d\t", current);

        for (int i = 0; i < n; i++) {
            if (adj[current][i] && !visited[i]) {
                visited[i] = 1;
                queue[rear++] = i;
            }
        }
    }
}

int main() {
    int v;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        visited[i] = 0;
    }

    printf("Enter graph data in matrix form:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);

    printf("Enter the starting vertex: ");
    scanf("%d", &v);
    bfs(v);

    for (i = 0; i < n; i++) {
        if (!visited[i]) {
            printf("\nBFS is not possible. Not all nodes are reachable.\n");
            return 0;
        }
    }

    return 0;
}
```

Output:

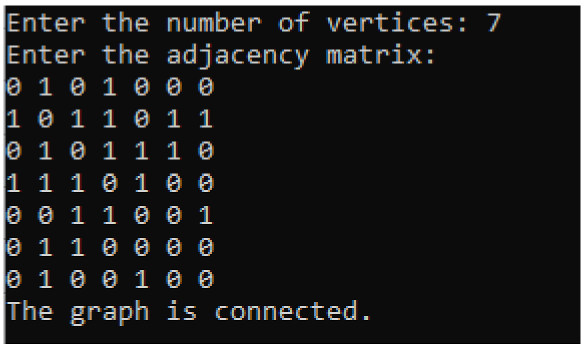


```
Enter the number of vertices: 7
Enter graph data in matrix form:
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
Enter the starting vertex: 0
0       1       3       2       5       6       4
```

#dfs

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 10
void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int num_vertices, bool
visited[MAX_VERTICES], int vertex) {
    visited[vertex] = true;
    int i;
    for ( i = 0; i < num_vertices; ++i) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            dfs(graph, num_vertices, visited, i);
        }
    }
}
bool is_connected(int graph[MAX_VERTICES][MAX_VERTICES], int num_vertices) {
    bool visited[MAX_VERTICES] = {false};
    dfs(graph, num_vertices, visited, 0);
    int i;
    for ( i = 0; i < num_vertices; ++i) {
        if (!visited[i]) {
            return false;
        }
    }
    return true;
}
int main() {
    int num_vertices;
    printf("Enter the number of vertices: ");
    scanf("%d", &num_vertices);
    int i,j;
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix:\n");
    for ( i = 0; i < num_vertices; ++i) {
        for (j = 0; j < num_vertices; ++j) {
            scanf("%d", &graph[i][j]);
        }
    }
    if (is_connected(graph, num_vertices)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

OUTPUT:



```
Enter the number of vertices: 7
Enter the adjacency matrix:
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
The graph is connected.
```

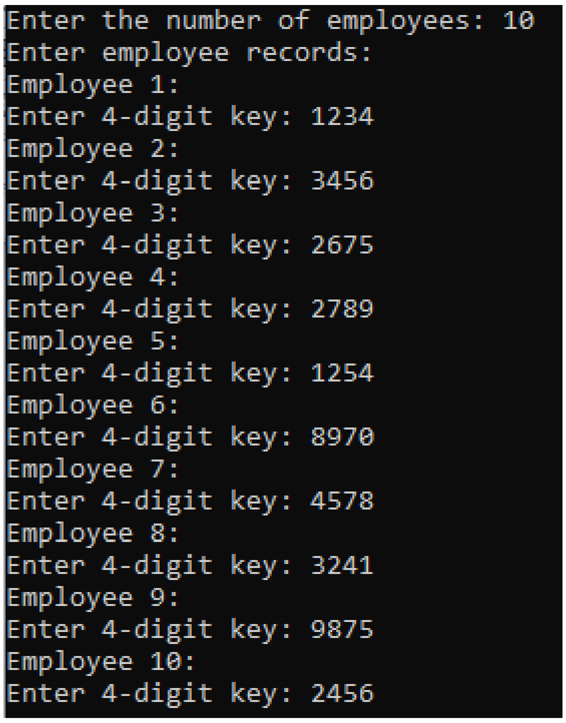
Lab Program 10:

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 -&gt; L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L.Resolve the collision (if any) using linear probing.

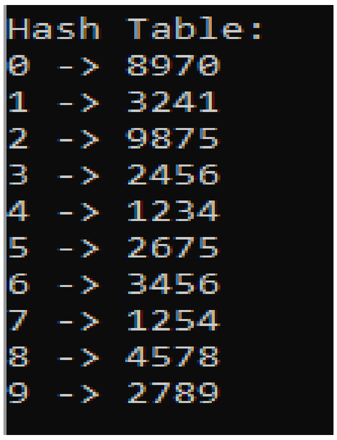
```
#include <stdio.h>
#include <stdlib.h>

#define MAX_EMPLOYEES 100
#define HASH_TABLE_SIZE 10
struct Employee {
   int key;
};
   int hashFunction(int key);
void insertEmployee(struct Employee employees[], int hashTable[], struct Employee emp);
void displayHashTable(int hashTable[]);

int main() {
   struct Employee employees[MAX_EMPLOYEES];
   int hashTable[HASH_TABLE_SIZE] = {0};
   int n, m, i;
   printf("Enter the number of employees: ");
   scanf("%d", &n);
   printf("Enter employee records:\n");
   for (i = 0; i < n; ++i) {
      printf("Employee %d:\n", i + 1);
      printf("Enter 4-digit key: ");
      scanf("%d", &employees[i].key);
      insertEmployee(employees, hashTable, employees[i]);
   }
   printf("\nHash Table:\n");
   displayHashTable(hashTable);

   return 0;
}
int hashFunction(int key) {
   return key % HASH_TABLE_SIZE;
}
void insertEmployee(struct Employee employees[], int hashTable[], struct Employee emp) {
   int index = hashFunction(emp.key);
   while (hashTable[index] != 0) {
      index = (index + 1) % HASH_TABLE_SIZE;
   }
   hashTable[index] = emp.key;
}
```

```
void displayHashTable(int hashTable[]) {
    int i;
    for (i = 0; i < HASH_TABLE_SIZE; ++i) {
        printf("%d -> ", i);
        if (hashTable[i] == 0) {
            printf("Empty\n");
        } else {
            printf("%d\n", hashTable[i]);
        }
    }
}
void displayProbingNumbers(int hashTable[]) {
    int i;
    for (i = 0; i < HASH_TABLE_SIZE; ++i) {
        if (hashTable[i] == 0) {
            printf("%d -> Empty\n", i);
        } else {
            printf("%d -> %d\n", i, i - hashFunction(hashTable[i]));
        }
    }
}
```

OUTPUT:



```
Enter the number of employees: 10
Enter employee records:
Employee 1:
Enter 4-digit key: 1234
Employee 2:
Enter 4-digit key: 3456
Employee 3:
Enter 4-digit key: 2675
Employee 4:
Enter 4-digit key: 2789
Employee 5:
Enter 4-digit key: 1254
Employee 6:
Enter 4-digit key: 8970
Employee 7:
Enter 4-digit key: 4578
Employee 8:
Enter 4-digit key: 3241
Employee 9:
Enter 4-digit key: 9875
Employee 10:
Enter 4-digit key: 2456
```



```
Hash Table:
0 -> 8970
1 -> 3241
2 -> 9875
3 -> 2456
4 -> 1234
5 -> 2675
6 -> 3456
7 -> 1254
8 -> 4578
9 -> 2789
```