VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

PRANEETA M REDDY (1BM22CS205)

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 PRAEETA M REDDY **(1BM22CS205)**, 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
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

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

Lab program 1:

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

```
#include <stdio.h>
#include <stdlib.h>
#define max 5
int s[max];
int top=-1;
void push(int value)
{
   if(top==max-1)
   {
      printf("overflow,cant push");

   }
   else{
      s[++top]=value;
   }
}
void pop()
{
   int value;
   if(top==-1){
      printf("stack underflow,cant pop\n");

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

   }
}
int isEmpty()
{
   if(top==-1)
      return 1;
   return 0;
}
int isFull()
{
   if(top==max-1)
      return 1;
   return 0;
}
void display()
{
```

```
    if(isEmpty())
    {
        printf("stack is empty\n");

    }
    else{
        printf("stack elements are:");
        for(int i=0;i<=top;i++)
        {
            printf("%d\t",s[i]);
        }
    }
}
int main()
{
    int choice;
    int n;
    while(1)
    {
      printf("\nenter 1 for push 2 for pop 3 for display 4 for exit\n");
      printf("enter ur choice:");
      scanf("%d",&choice);
      switch(choice)
      {
       case 1:printf("enter the value:");
              scanf("%d",&n);
              push(n);
              break;
       case 2:pop();
              break;
       case 3:display();
              break;
       case 4:exit(0);
       default:printf("invalid input");

      }

    }
return 0;
}
```

Output:

```
enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:1
enter the value:10

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:1
enter the value:20

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:1
enter the value:30

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:1
enter the value:40

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:1
enter the value:50

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:1
enter the value:60
overflow,cant push
enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:3
stack elements are:10   20      30      40      50
enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:2
50 is popped

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:2
40 is popped
```

```
enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:2
30 is popped

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:2
20 is popped

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:2
10 is popped

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur choice:2
stack underflow,cant pop

enter 1 for push 2 for pop 3 for display 4 for exit
enter ur 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>
#include <ctype.h>
#define max 100
char s[max];
int top=-1;
void push(char value)
{
          s[++top]=value;
         }
char pop()
{
   char value=' ';
   if (top==-1)
      return value;
   else
      return s[top--];
}
int pre(char value)
{
   if(value=='^')
   {
      return 3;
   }
   else if(value =='*' || value =='/')
   {
      return 2;
   }
   else if(value =='+' || value =='-')
   {
      return 1;
   }
   else
   {
      return 0;
   }

}
void convert(char infix[],char postfix[])
{
   int i=0,j=0;
   char x;
   while(infix[i]!='\0')
```

```
    {
        if(infix[i]=='(')
        {
            push(infix[i]);
            i++;
        }
        else if(infix[i]==')')
        {
            while((s[top]!='(') && (top!=-1))
            {
                postfix[j++]=pop();

            }
            x=pop();
            i++;
        }
        else if(isalnum(infix[i]))
        {
            postfix[j++]=infix[i];
            i++;

        }
        else
        {
            while((s[top]!='(') &&( top!=-1) && (pre(s[top])>=pre(infix[i])))
            {
                postfix[j++]=pop();
            }
            push(infix[i]);
            i++;
        }
    }
    while((s[top]!='(') && (top!=-1))
    {
        postfix[j++]=pop();
    }
    postfix[j]='\0';
}
void main()
{
    char infix[100],postfix[100];
    printf("enter the exp:");
    scanf("%s",infix);
    convert(infix,postfix);
    printf("postfix:");
    printf("%s",postfix);
}
```

Output:

```
enter the exp:((a+b)*c-d)
postfix:ab+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:

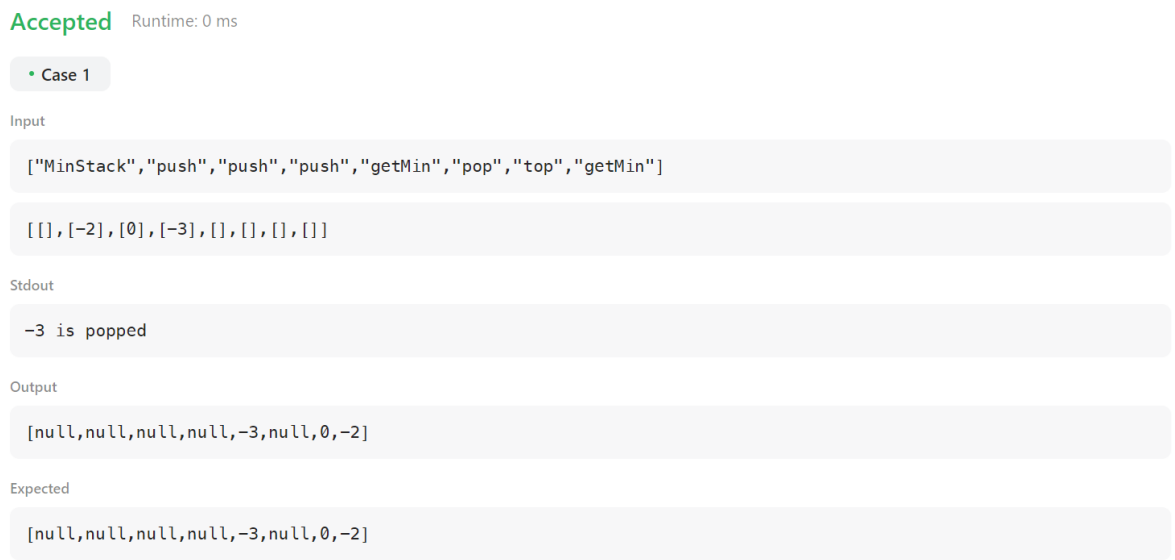
MinStack() initializes the stack object.
void push(int val) pushes the element val onto the stack.
void pop() removes the element on the top of the stack.
int top() gets the top element of the stack.
int getMin() retrieves the minimum element in the stack.
You must implement a solution with O(1) time 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:



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 4
int rear=-1;
int front=-1;
int q[max];
void enqueue(int x)
{
   if (rear==max-1)
   {
      printf("queue overflow\n");
   }
   else if(front==-1 && rear==-1)
   {
      front=rear=0;
   }
   else{
      rear++;
   }
   q[rear]=x;
}
int dequeue()
{
   int x=-1;
   if (front==-1 || front>rear)
   {
      printf("\nunderflow");
      return -1;
   }
   else
   {
      x=q[front];
      front++;
      if(front>rear)
         front=rear=-1;
      return x;

   }
}
void display()
{
   if(front==-1 || front>rear)
   {
```

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

    }
}
void main()
{
    int c,no,x;
    while(1)
    {
        printf("enter 1 for insert 2 for delete 3 for display 4 for exit\n");
        printf("enter the choice:");
        scanf("%d",&c);
        switch (c)
        {
        case 1:
            printf("enter the no:");
            scanf("%d",&no);
            enqueue(no);

            break;
        case 2:x=dequeue();
                if (x!=-1)
                {
                    printf("%d is popped\n",x);
                }
                break;
        case 3:display();
                break;
        case 4:exit(0);
                // break;


        default:printf("invalid\n");
            break;
    }
}
}
```

Output:

```
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:20
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:30
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:40
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:50
queue overflow
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:3
10      20      30      50      enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
10 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
20 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
30 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
50 is popped
```

```
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2

underflowenter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:4
```

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 10
int rear=-1;
int front=-1;
int q[max];
int isfull()
{
   if(front==rear+1 || front==0 && rear==max-1)
      return 1;
   return 0;
 }
int is_empty()
{
   if(front==-1 && rear==-1)
```

```
        return 1;
    return 0;
}
void enqueue(int x)
{
    if(isfull())
    {
        printf("overflow\t");
    }
    else if(front==-1 && rear==-1)
    {
        front=0;
        rear=0;
    }
    else
    {
        rear=(rear+1)%max;
    }
    q[rear]=x;
}
int dequeue()
{
    int vlaue=-1;
    if(is_empty())
    {
        printf("underflow\t");
        return -1;
    }
    else
   {
        vlaue=q[front];
    if(front==rear)
    {
        front=-1;
        rear=-1;
    }
    else{
        front=(front+1)%max;
    }
    return vlaue;
   }
}
void display()
{
    int i;
    if(is_empty())
    {
        printf("underflow\t");

    }
```

```
    else{
        printf("elements are:");
        for( i=front;i!=rear;i=(i+1)%max)
        {
            printf("%d\t",q[i]);
        }
        printf("%d",q[i]);
    }
}
void main()
{
    int c,no,x;

    while(1)
    {
        printf("enter 1 for insert 2 for delete 3 for display 4 for exit\n");
        printf("enter the choice:");
        scanf("%d",&c);
        switch (c)
        {
        case 1:
            printf("enter the no:");
            scanf("%d",&no);
            enqueue(no);

            break;
        case 2:x=dequeue();
                if (x!=-1)
                {
                    printf("%d is popped\n",x);
                }
                break;
        case 3:display();
                break;
        case 4:exit(0);
                // break;


        default:printf("invalid\n");
            break;
    }
}
}
```

Output:

```
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:20
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:11
invalid
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:30
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:3
elements are:10 20      30enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
10 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
20 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
30 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
underflow       enter 1 for insert 2 for delete 3 for display 4 for exit
enter the 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;
}*first=NULL;
void insert(struct node  **p,int pos,int no)
{
   struct node *t;
   t=(struct node*) malloc (sizeof(struct node));
   t->data=no;
   if(pos==0)
   {
      t->next=*p;
      *p=t;
   }
   else{
      struct node *temp=*p;
      for(int i=1;i<pos && (p!=NULL);i++)
         temp=temp->next;
      t->next=temp->next;
      temp->next=t;

   }

}
void display(struct node *p)
{
   while(p!=NULL)
   {
      printf("%d\t",p->data);
      p=p->next;
   }
}
void main()
{
   int ch,p,n;
   while(1)
   {
      printf("enter 1 to insert 2 to display 3 to exit");
```

```
        printf("enter the choice:");
        scanf("%d",&ch);
        switch(ch)
        {
           case 1:printf("enter the  pos and no where to insert:");
                  scanf("%d%d",&p,&n);
                  insert(&first,p,n);
                  break;
           case 2:display(first);
                  break;
           case 3:exit(0);
           default:printf("invalid choice");
                  break;

        }
    }
}
```

Output:

```
enter 1 to insert 2 to display 3 to exitenter the choice:1
enter the  pos and no where to insert:0 10
enter 1 to insert 2 to display 3 to exitenter the choice:2
10      enter 1 to insert 2 to display 3 to exitenter the choice:1
enter the  pos and no where to insert:0 20
enter 1 to insert 2 to display 3 to exitenter the choice:2
20      10      enter 1 to insert 2 to display 3 to exitenter the choice:1
enter the  pos and no where to insert:1 30
enter 1 to insert 2 to display 3 to exitenter the choice:2
20      30      10      enter 1 to insert 2 to display 3 to exitenter the choice:3
```

4 b)Program - Leetcode platform-Reverse Linked List II

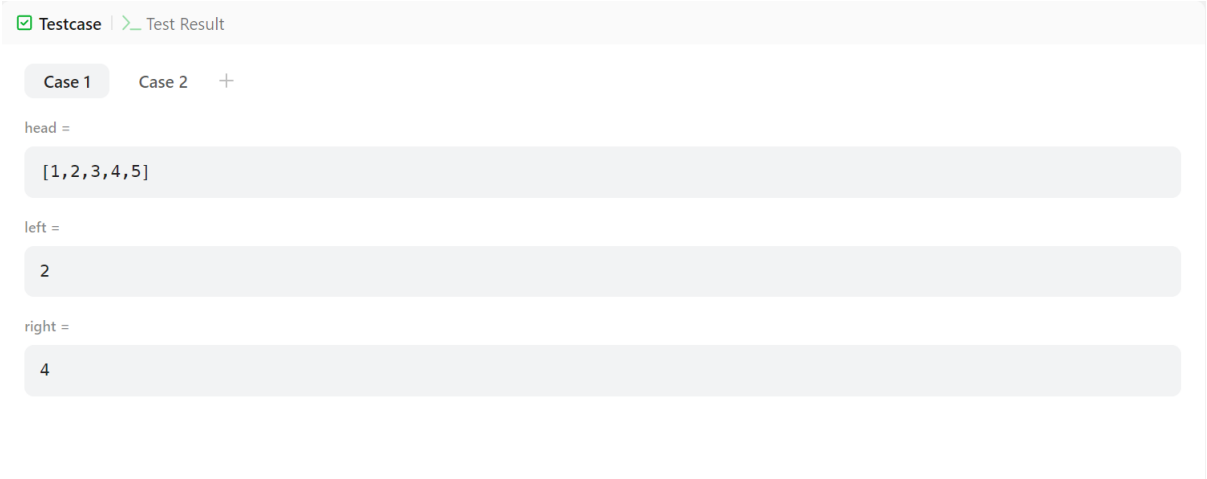
Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list.

```
/**
 * 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;
```

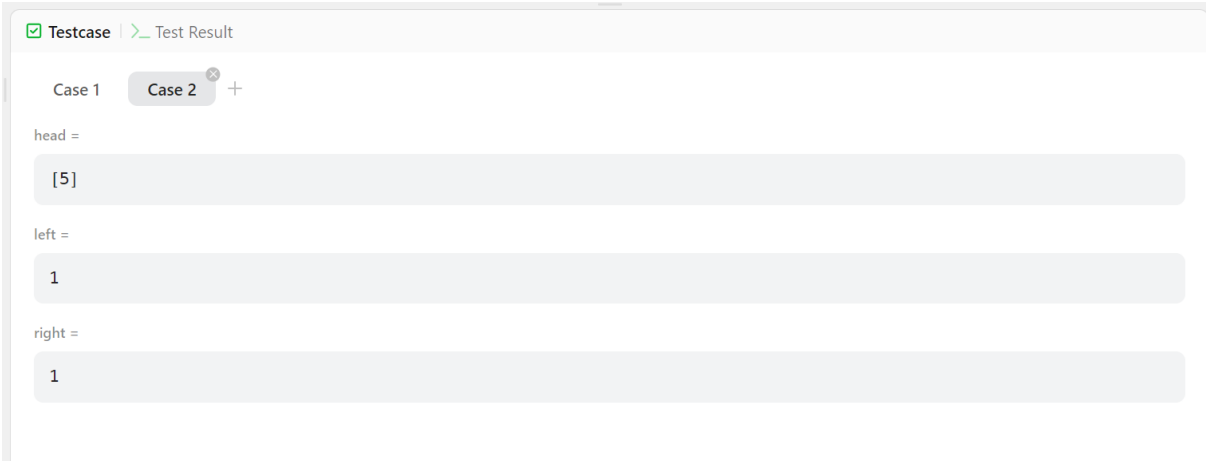
```
    for (int i = 1; i < left; i++) {
        ptr1= ptr;
        ptr = ptr->next;
    }

    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:



Test case2:



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:

Test case2:

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;
} *top = NULL;
int empty()
{
   if (top == NULL)
      return 1;
   return 0;
}
int full()
{
   struct node *t;
   t = (struct node*)malloc(sizeof(struct node));
   if (t == NULL)
      return 1;
   return 0;
}
void push(int x)
{
   struct node *t;
   t = (struct node*)malloc(sizeof(struct node));

   if (full())
   {
      printf("overflow");
   }
   else
   {
      t->data = x;
      t->next = top;
      top = t;
   }
}
int pop()
{
   struct node *t;
   // t = (*struct node)malloc(sizeof(struct node));
   int x = -1;
   if(empty())
   {
      printf("stack underflow");
      return x;
   }
```

```
    else
    {
        t = top;
        top = top->next;
        x = t->data;
        free(t);
        return x;
    }
}
void display()
{
    struct node *t;
    // t = (*struct node)malloc(sizeof(struct node));
    t=top;
    while(t!=NULL)
    {
        printf("%d\t",t->data);
        t=t->next;
    }
    printf("\n");

}
void main()
{
    int c, no, x;
    while (1)
    {
        printf("enter 1 for insert 2 for delete 3 for display 4 for exit\n");
        printf("enter the choice:");
        scanf("%d", &c);
        switch (c)
        {
        case 1:
            printf("enter the no:");
            scanf("%d", &no);
            push(no);

            break;
        case 2:
            x = pop();
            if (x != -1)
            {
                printf("%d is popped\n", x);
            }
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
```

```
        default:
            printf("invalid\n");
            break;
        }
    }
}
```

Output:

```
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:3
10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:20
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:30
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:3
30      20      10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
30 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:4
```

//Queue

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
} *top = NULL;
int empty()
{
    if (top == NULL)
        return 1;
    return 0;
}
int full()
{
    struct node *t;
    t = (struct node*)malloc(sizeof(struct node));
    if (t == NULL)
        return 1;
    return 0;
}
void push(int x)
{
    struct node *t;
    t = (struct node*)malloc(sizeof(struct node));
```

```
    if (full())
    {
        printf("overflow");
    }
    else
    {
        t->data = x;
        t->next = top;
        top = t;
    }
}
int pop()
{
    struct node *t;
    // t = (*struct node)malloc(sizeof(struct node));
    int x = -1;
    if(empty())
    {
        printf("stack underflow");
        return x;
    }
    else
    {
        t = top;
        top = top->next;
        x = t->data;
        free(t);
        return x;
    }
}
void display()
{
    struct node *t;
    // t = (*struct node)malloc(sizeof(struct node));
    t=top;
    while(t!=NULL)
    {
        printf("%d\t",t->data);
        t=t->next;
    }
    printf("\n");

}
void main()
{
    int c, no, x;
    while (1)
    {
        printf("enter 1 for insert 2 for delete 3 for display 4 for exit\n");
```

```
        printf("enter the choice:");
        scanf("%d", &c);
        switch (c)
        {
        case 1:
            printf("enter the no:");
            scanf("%d", &no);
            push(no);

            break;
        case 2:
            x = pop();
            if (x != -1)
            {
                printf("%d is popped\n", x);
            }
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);

        default:
            printf("invalid\n");
            break;
        }
    }
}
```

Output:

```
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:1
enter the no:20
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:3
10      20
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:2
10 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice:4
```

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;
}*first=NULL;
void create(int a[],int n){
   struct node *t,*last;
   first=(struct node *) malloc (sizeof(struct node));
   first->data=a[0];
   first->prev=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=last->next;
      t->prev=last;
      last->next=t;
      last=t;
   }
}
void insertleft(int val,int pos){
   struct node *t,*ptr;
   int i,loc;
   loc=pos;
   t=(struct node *) malloc (sizeof(struct node));
   if(t==NULL){
      printf("overflow");
   }
   t->data=val;
   if(pos==1){
         t->prev=NULL;
         t->next=first;


         if(first!=NULL){
            first->prev=t;
         }
         first=t;
```

```
    }

    else
    {
            ptr=first;
            for(int i=0;i<loc-2;i++){
                ptr=ptr->next;
            }
            t->next=ptr->next;
            t->prev=ptr;

            if(ptr!=NULL){
                ptr->next->prev=t;
            }
             ptr->next=t;
    }
    printf("node inserted ");

}
void delevalue(int val){
    struct node *ptr;
    int value;
    value=val;
    ptr=first;
    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==first){
                first=ptr->next;
            }
            free(ptr);
        printf("value %d deleted",value);
        return;
        }
        ptr=ptr->next;

    }
    printf("%d value not found",value);

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

```
        p=p->next;
    }
    printf("\n");
}
void main(){
    int a[10],n;
    int val,key,loc;
    printf("read n");
    scanf("%d",&n);
    printf("enter the values:");
    for(int i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    create(a,n);
    display(first);
    printf("enter the value to be inserted:");
    scanf("%d",&val);
    printf("enter the loc  to be inserted at:");
    scanf("%d",&loc);
    insertleft(val,loc);
    display(first);
    printf("enter the key element to be deleted");
    scanf("%d",&key);
    delevalue(key);
    display(first);
}
```

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 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 data;
   struct node *left;
   struct node *right;
   }*root=NULL;
struct node *create(struct node *t,int ele){

   if(t==NULL){
      struct node *temp=(struct node *) malloc (sizeof(struct node));
      temp->data=ele;
      temp->left=temp->right=NULL;
      return temp;
   }
   else{
         if(ele<t->data){
            t->left=create(t->left,ele);
         }
         else{
            t->right=create(t->right,ele);
         }

   }
   return t;
}
void pre(struct node *root){
   struct node *r;
   r=root;
   if(r!=NULL){
      printf("%d\t",r->data);
      pre(r->left);
      pre(r->right);
   }
}
void in(struct node *root){
   struct node *r;
   r=root;
   if(r!=NULL){

      in(r->left);
```

```
        printf("%d\t",r->data);
        in(r->right);
    }
}

void post(struct node *root){
    struct node *r;
    r=root;
    if(r!=NULL){

        post(r->left);
        post(r->right);
        printf("%d\t",r->data);
    }
}
void main(){
    int n,ele;
    printf("enter the no of elements:");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        printf("enter the element %d:",i+1);
        scanf("%d",&ele);
        root=create(root,ele);
    }
    printf("display the elements in preorder traversal:");
    pre(root);
    printf("\ndisplay the elements in inorder traversal:");
    in(root);
    printf("\ndisplay the elements in postorder traversal:");
    post(root);

}
```

Output:

```
enter the no of elements:10
enter the element 1:20
enter the element 2:10
enter the element 3:5
enter the element 4:15
enter the element 5:25
enter the element 6:22
enter the element 7:50
enter the element 8:18
enter the element 9:40
enter the element 10:70
display the elements in preorder traversal:20   10      5       15      18      25      22      50      40      70
display the elements in inorder traversal:5     10      15      18      20      22      25      40      50      70
display the elements in postorder traversal:5   18      15      10      22      40      70      50      25      20
```

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 <bits/stdc++.h>
using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'swapNodes' function below.
 *
 * The function is expected to return a 2D_INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. 2D_INTEGER_ARRAY indexes
 *  2. INTEGER_ARRAY queries
 */
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

// Function to perform in-order traversal of the binary tree
void inOrderTraversal(TreeNode* root, vector<int>& result) {
    if (root == nullptr)
        return;

    inOrderTraversal(root->left, result);
    result.push_back(root->data);
    inOrderTraversal(root->right, result);
}

// Function to swap subtrees at specified depths
void swapSubtrees(TreeNode* root, int k, int depth) {
```

```
    if (root == nullptr)
        return;

    if (depth % k == 0) {
        TreeNode* temp = root->left;
        root->left = root->right;
        root->right = temp;
    }

    swapSubtrees(root->left, k, depth + 1);
    swapSubtrees(root->right, k, depth + 1);
}

// Function to build the binary tree from the given indexes
TreeNode* buildTree(vector<vector<int>>& indexes) {
    queue<TreeNode*> nodes;
    TreeNode* root = new TreeNode(1);
    nodes.push(root);

    for (auto& idx : indexes) {
        TreeNode* curr = nodes.front();
        nodes.pop();

        if (idx[0] != -1) {
            curr->left = new TreeNode(idx[0]);
            nodes.push(curr->left);
        }

        if (idx[1] != -1) {
            curr->right = new TreeNode(idx[1]);
            nodes.push(curr->right);
        }
    }

    return root;
}
vector<vector<int>> swapNodes(vector<vector<int>> indexes, vector<int> queries) {
vector<vector<int>> result;
    TreeNode* root = buildTree(indexes);

    for (int k : queries) {
        swapSubtrees(root, k, 1);
        vector<int> traversal;
        inOrderTraversal(root, traversal);
        result.push_back(traversal);
    }

    return result;
}
```

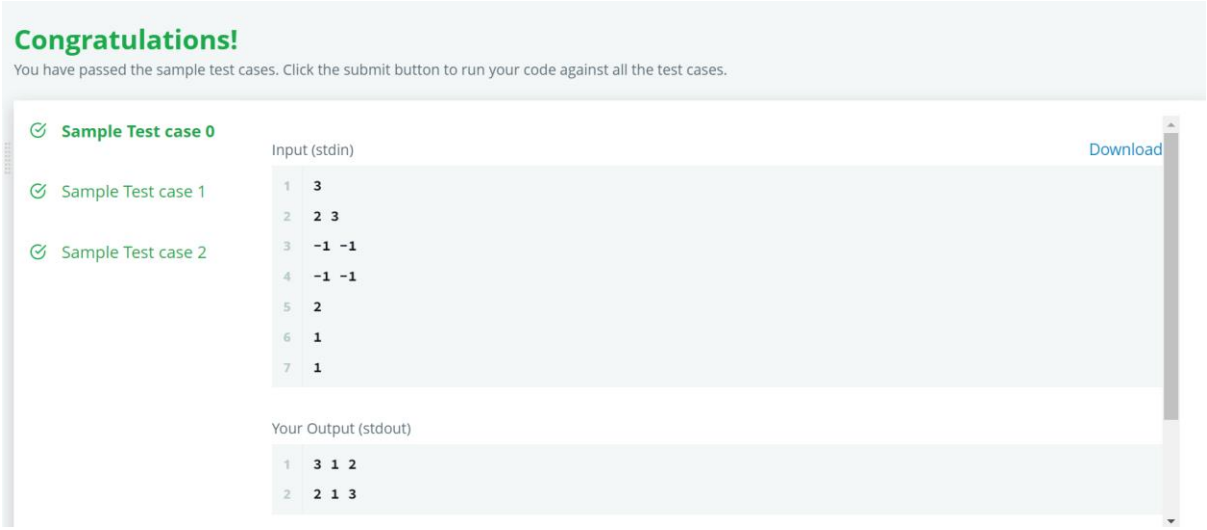
```
int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    vector<vector<int>> indexes(n);

    for (int i = 0; i < n; i++) {
        indexes[i].resize(2);

        string indexes_row_temp_temp;
        getline(cin, indexes_row_temp_temp);

        vector<string> indexes_row_temp = split(rtrim(indexes_row_temp_temp));

        for (int j = 0; j < 2; j++) {
            int indexes_row_item = stoi(indexes_row_temp[j]);

            indexes[i][j] = indexes_row_item;
        }
    }

    string queries_count_temp;
    getline(cin, queries_count_temp);

    int queries_count = stoi(ltrim(rtrim(queries_count_temp)));

    vector<int> queries(queries_count);

    for (int i = 0; i < queries_count; i++) {
        string queries_item_temp;
        getline(cin, queries_item_temp);

        int queries_item = stoi(ltrim(rtrim(queries_item_temp)));

        queries[i] = queries_item;
    }

    vector<vector<int>> result = swapNodes(indexes, queries);

    for (size_t i = 0; i < result.size(); i++) {
        for (size_t j = 0; j < result[i].size(); j++) {
            fout << result[i][j];

            if (j != result[i].size() - 1) {
                fout << " ";
```

```
            }
        }

        if (i != result.size() - 1) {
            fout << "\n";
        }
    }

    fout << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));
```

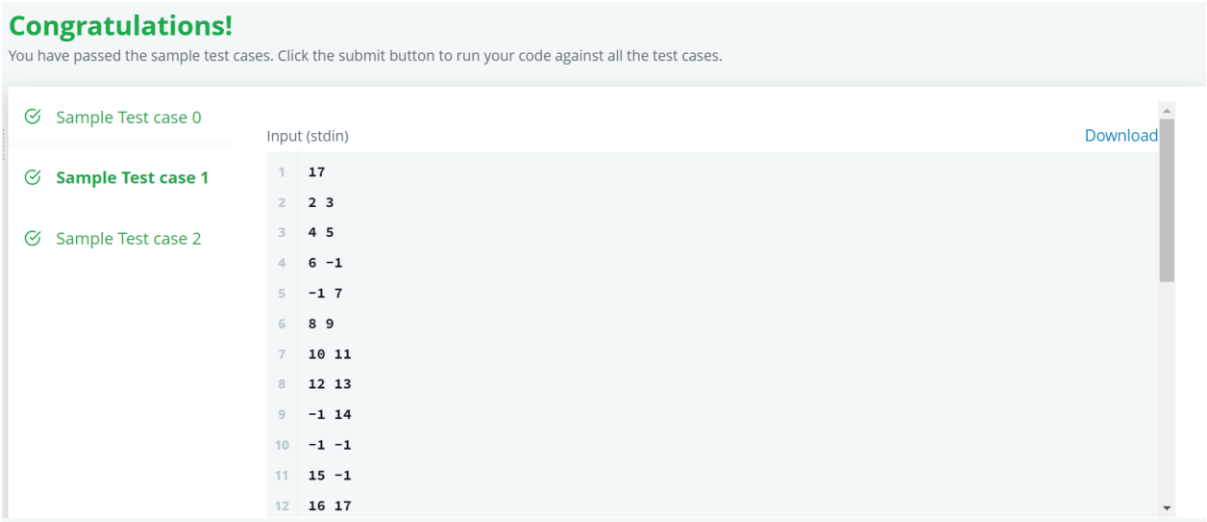
```
    return tokens;
}
```

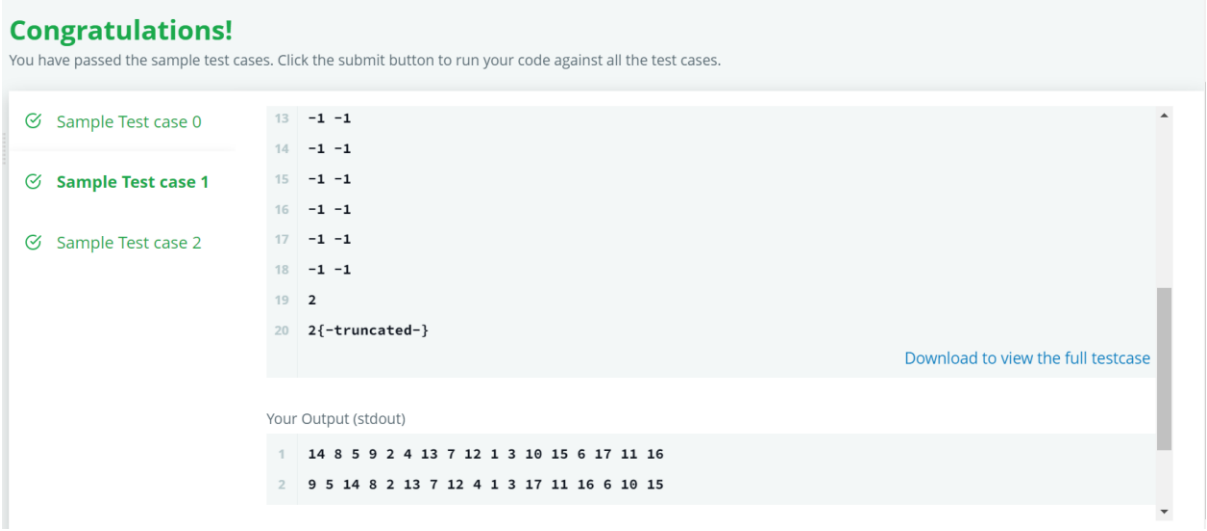
Output:

Test case 1:

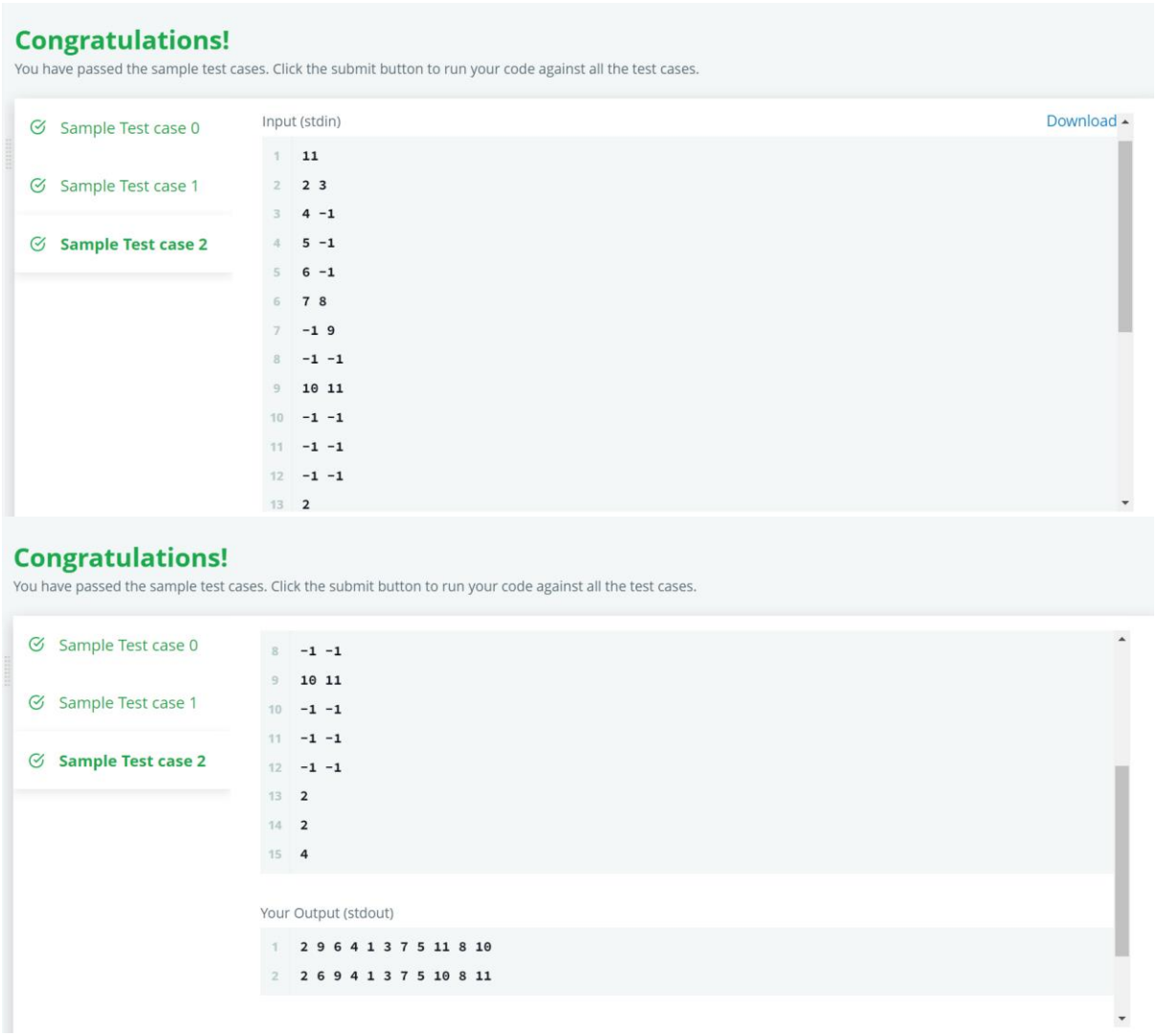


Test case 2:





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.
Sample Test case 0
Sample Test case 1
Sample Test case 2
-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>
#include <stdlib.h>
struct node{
   int data;
   struct node *next;
}*front=NULL,*rear=NULL;
void enquque(int x){
   struct node *t=(struct node*) malloc(sizeof(struct node));
   if(t==NULL){
      printf("queue is overflow");
   }
   else{
         t->data=x;
         t->next=NULL;
   if(front==NULL){
      front=rear=t;


   }
   else{
      rear->next=t;
      rear=t;
   }
   }
}
int dequque(){
   struct node *t;
   int x=-1;
   if(front==NULL){
      printf("queque is empty");
      return x;
   }
   else{
         t=front;
      x=t->data;
      front=front->next;
   free(t);
   return x;

   }
}
int isempt(){
```

```
    if(front==NULL){
        return 1;
    }
    return 0;
}

//BFS method
void bfs(int i,int visited[],int a[][20],int n){
    int u;
    printf("bfs traversal:");
    printf("%d ",i);
    visited[i-1]=1;
    enquque(i-1);
    while(!isempt()){
            u=dequque();
            for(int v=0;v<n;v++){
                if(a[u][v]==1 && visited[v]==0){
                    printf("%d ",v+1);
                    visited[v]=1;
                    enquque(v);
                }
            }

    }
}
//DFS method
void dfs(int i,int visited[],int a[][20],int n){
    if(visited[i-1]==0){
        printf("%d ",i);
        visited[i-1]=1;
        for(int j=0;j<n;j++){
            if(a[i-1][j]==1 && visited[j]==0){
                dfs(j+1,visited,a,n);
            }
        }
    }
}
void main(){
    int visited[20]={0};
    int a[20][20];
    int n,first;
    int count=0;
    printf("enter the number of vertices:");
    scanf("%d",&n);
    printf("enter the adjacency matrix:");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d",&a[i][j]);
        }
    }
```

```
    printf("the adjacency matrix:\n");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            printf("%d\t",a[i][j]);
        }
        printf("\n");
    }
    printf("enter the starting vertex: ");
    scanf("%d",&first);
    bfs(first,visited,a,n);
    for(int i=0;i<20;i++){
        visited[i]=0;
    }
    printf("\ndfs traversal:");
    dfs(first,visited,a,n);
    for(int i=0;i<n;i++){
        if(visited[i]==1){
            count++;
        }
    }
//DFS to check whether a graph is connected or not
    if(count==n){
        printf("\ngraph is connected");
    }
    else{
        printf("\ngraph is not connected");
    }
}
```

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 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
enter the starting vertex: 4
bfs traversal:4 1 2 3 5 6 7
dfs traversal:4 1 2 3 5 7 6
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 -> 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 SIZE 10
struct Employee {
   int key;
};
int hash(int key) {
   return key % SIZE;
}
int probe(int H[], int key) {
   int index = hash(key);
   int i = 0;
   while (H[(index + i) % SIZE] != 0)
      i++;
   return (index + i) % SIZE;
}
void Insert(int H[], int key) {
   int index = hash(key);
   if (H[index] != 0)
      index = probe(H, key);
   if (index == -1) {
      printf("Error: Hash table is full. Unable to insert employee ID %d\n", key);
      return;
   }
   H[index] = key;
    printf("Employee ID %d inserted at index %d\n", key, index);
}
int Search(int H[], int key) {
   int index = hash(key);
   int i = 0;
   while (H[(index + i) % SIZE] != key) {
      if (H[(index + i) % SIZE] == 0)
         return -1;
      i++;
   }
   return (index + i) % SIZE;
}
int main() {
   int HT[SIZE] = {0};
   // Sample employee records
```

```
    Insert(HT, 1234);
    Insert(HT, 5678);
    Insert(HT, 9012);
    Insert(HT, 3456);
     Insert(HT, 3446);
     Insert(HT, 3458);
    int searchKey = 9012; // Key to search for
    int resultIndex = Search(HT, searchKey);
    if (resultIndex != -1) {
        printf("Employee with key %d found at index %d!\n", searchKey, resultIndex);
    } else {
        printf("Employee with key %d not found!\n", searchKey);
    }
   return 0;
}
```

```
Employee ID 1234 inserted at index 4
Employee ID 5678 inserted at index 8
Employee ID 9012 inserted at index 2
Employee ID 3456 inserted at index 6
Employee ID 3446 inserted at index 7
Employee ID 3458 inserted at index 9
Employee with key 9012 found at index 2!
```