

# **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



## **LAB REPORT**

**On**

### **DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**PIYUSH BELLUBBI (1BM22CS192)**

**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 PIYUSH BELLUBBI (**1BM22CS192**), 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

Sl. No.	Experiment Title	Page No.
1	Stack implementation	4
2	Postfix expression to Infix expression	6
3	Implementation of queue and circular queue	12
4	Demonstrate insertion in singly linked list	17
5	Demonstrate deletion in singly linked list	22
6	Operations on singly linked list . Implement stack and queue using singly linked list.	28
7	Implementation of doubly linked list	37
8	Implementation of binary search tree	43
9	Traversal of graph using BFS method. Check if graph is connected using DFS method	47
10	Implement hash table and resolve collisions using linear probing.	56

#### 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>
#define N 5
void Push();
void Pop();
void Peek();
void Display();

int stack[N];
int top=-1;
void main()
{
    int ch;
    do{
        printf("\nEnter choice: 1.Push 2.Pop 3.Peek 4.Display \n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:Push();
            break;
            case 2:Pop();
            break;
            case 3:Peek();
            break;
            case 4:Display();
            break;
            default:printf("Invalid choice");
        }
    }
    while(ch!=0);
    getch();
}

void Push()
{
    int x;
    printf("Enter data: \n");
    scanf("%d",&x);
    if(top==N-1)
    {
        printf("Overflow");
    }
    else
    {
        top++;
        stack[top]=x;
    }
}

void Pop()
{
    int item;
```

```

        if(top== -1)
        {
            printf("Underflow");
        }
        else
        {
            top--;
            printf("\nTop after popping: %d",stack[top]);
        }
    }
void Peek()
{
    if(top== -1)
    {
        printf("Element does not exist.");
    }
    else
    {
        printf("%d",stack[top]);
    }
}
void Display()
{
    int i;
    for(i=top;i>=0;i--)
    {
        printf("\nStack is:\n%d",stack[i]);
    }
}

```

### Output:

```
C:\Users\Admin\Desktop\1BM22CS192(ds)\prg3stackswitch.exe

Enter choice: 1.Push 2.Pop 3.Peek 4.Display
1
Enter data:
3

Enter choice: 1.Push 2.Pop 3.Peek 4.Display
1
Enter data:
4

Enter choice: 1.Push 2.Pop 3.Peek 4.Display
1
Enter data:
5

Enter choice: 1.Push 2.Pop 3.Peek 4.Display
2

Top after popping: 4
Enter choice: 1.Push 2.Pop 3.Peek 4.Display
4

Stack is:
4
Stack is:
3
Enter choice: 1.Push 2.Pop 3.Peek 4.Display
```

## Lab program 2:

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

```
#include<stdio.h>

char stack[20];
int top=-1;

void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if(top==-1)
        return -1;
    else
        return stack[top--];
}

int precedence(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
}

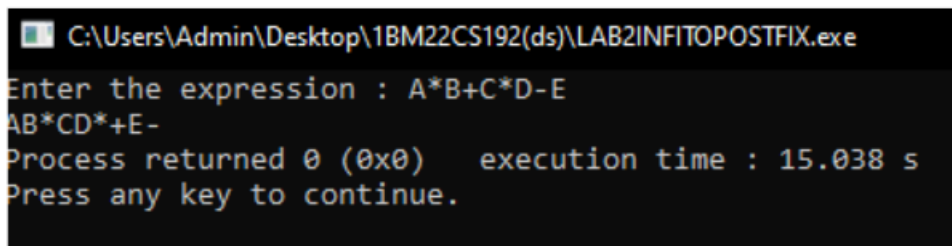
int main()
{
    char exp[20];
    char *e,x;
    printf("Enter the expression : ");
    scanf("%s",exp);
    e = exp;
    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
                printf("%c",x);
        }
        e++;
    }
}
```

```

        else
        {
            while(precedence(stack[top]) >= precedence(*e))
                printf("%c",pop());
            push(*e);
        }
        e++;
    }
    while(top != -1)
    {
        printf("%c",pop());
    }
}

```

## OUTPUT:



```

C:\Users\Admin\Desktop\1BM22CS192(ds)\LAB2INFITOPSTFIX.exe
Enter the expression : A*B+C*D-E
AB*CD*+E-
Process returned 0 (0x0)   execution time : 15.038 s
Press any key to continue.

```

## POSTFIX EVALUATON:

```

#include<stdio.h>
int stack[20];
int top = -1;

void push(int x)
{
    stack[++top] = x;
}

int pop()
{
    return stack[top--];
}

int main()
{
    char exp[20];
    char *e;
    int n1,n2,n3,num;
    printf("Enter the expression :: ");
    scanf("%s",exp);
    e = exp;
    while(*e != '\0')
    {
        if(isdigit(*e))
        {
            num = *e - 48;
            push(num);
        }
    }
}

```

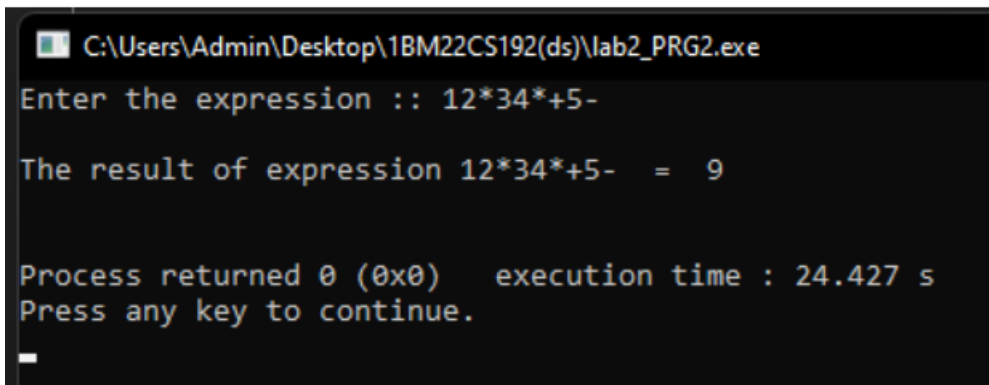


```

else
{
    n1 = pop();
    n2 = pop();
    switch(*e)
    {
        case '+': n3 = n1 + n2;
            break;
        case '-': n3 = n2 - n1;
            break;
        case '*': n3 = n1 * n2;
            break;
        case '/': n3 = n2 / n1;
            break;
    }
    push(n3);
}
e++;
}
printf("\nThe result of expression %s = %d\n\n",exp,pop());
return 0;
}

```

## OUTPUT:



```

C:\Users\Admin\Desktop\1BM22CS192(ds)\lab2_PRG2.exe
Enter the expression :: 12*34*+5-

The result of expression 12*34*+5- = 9

Process returned 0 (0x0)   execution time : 24.427 s
Press any key to continue.
_

```

### Lab program 3:

3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display

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

```
#include<stdio.h>
#define size 30

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

void insert(int a){
    if(rear==size-1){
        printf("Queue overflow\n");
        return;
    }
    else{
        if(front==-1)
            front=0;
        queue[++rear]=a;
    }
}

void delete(){
    if(front==-1||front>rear){
        printf("Queue Empty\n");
    }
    else{
        front++;
    }
}

void display(){
    if(front==-1){
        printf("Queue Empty\n");
        return;
    }
    printf("Queue:");
    for(int i=front; i<=rear; i++){
        printf("%d ", queue[i]);
    }
    printf("\n-----\n");
}

void main(){
    int choice;
    int a;
    while(1){
```

```
printf("Queue operations:\n1.Push\n2.Pop\n3.Display\nChoice:");
scanf("%d",&choice);

switch (choice)
{
case 1:
    printf("Enter Element:");
    scanf("%d",&a);
    insert(a);
    display();
    break;

case 2:
    delete();
    display();
    break;

case 3:
    display();
    break;
}
}
```

OUTPUTS:

```
1.Push
2.Pop
3.Display
Choice:1
Enter value:1
Queue:1
1.Push
2.Pop
3.Display
Choice:1
Enter value:4
Queue:1 4
1.Push
2.Pop
3.Display
Choice:
1
Enter value:6
Queue:1 4 6
1.Push
2.Pop
3.Display
Choice:2
Queue:4 6
1.Push
2.Pop
3.Display
Choice:_
```

**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>

#define SIZE 6

int CQ[SIZE];
int front = -1, rear = -1;

int checkFull ()
{
    if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))
    {
```

```

        return 1;
    }
    return 0;
}

int checkEmpty ()
{
    if (front == -1)
    {
        return 1;
    }
    return 0;
}

void enqueue (int value)
{
    if (checkFull ())
        printf ("Overflow condition\n");

    else
    {
        if (front == -1)
            front = 0;

        rear = (rear + 1) % SIZE;
        CQ[rear] = value;
        printf ("%d was enqueued to circular queue\n", value);
    }
}

int dequeue ()
{
    int variable;
    if (checkEmpty ())
    {
        printf ("Underflow condition\n");
        return -1;
    }
    else
    {
        variable = CQ[front];
        if (front == rear)
        {
            front = rear = -1;
        }
        else
        {
            front = (front + 1) % SIZE;
        }
        printf ("%d was dequeued from circular queue\n", variable);
        return 1;
    }
}

void display()
{
    int i;

```

```

        if (checkEmpty())
            printf("Nothing to dequeue\n");
        else
        {
            printf("\nThe queue looks like: \n");
            for(i=front;i!=rear;i=(i+1)%SIZE)
            {
                printf("%d \t",CQ[i]);
            }
            printf("%d \n\n",CQ[i]);
        }
    }
}
int main()
{
    dequeue();

    enqueue (15);
    enqueue (20);
    enqueue (25);
    enqueue (30);
    enqueue (35);

    display ();
    dequeue ();
    dequeue ();

    display ();

    enqueue (40);
    enqueue (45);
    enqueue (50);
    enqueue (55);
    display ();

    return 0;
}

```

OUTPUTS:

```
C:\Users\Admin\Desktop\1BM22CS192(ds)\Codes\LAB3_CircularQ.exe
Underflow condition
15 was enqueued to circular queue
20 was enqueued to circular queue
25 was enqueued to circular queue
30 was enqueued to circular queue
35 was enqueued to circular queue

The queue looks like:
15      20      25      30      35

15 was dequeued from circular queue
20 was dequeued from circular queue

The queue looks like:
25      30      35

40 was enqueued to circular queue
45 was enqueued to circular queue
50 was enqueued to circular queue
Overflow condition

The queue looks like:
25      30      35      40      45      50

Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
_
```

#### Lab program 4:

WAP to Implement Singly Linked List with following operations

- a. Create a linked list.
- b. Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

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

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

Node* head = NULL;

void push();
void append();
void insert();
void display();

int main() {
    int choice;
    while (1) {
        printf("1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Insert at position\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                append();
                break;
            case 3:
                insert();
                break;
            case 4:
                display();
                break;
            default:
                printf("Exiting the program");
                return 0;
        }
    }
}
```



```

void push() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
    temp->data = new_data;
    temp->next = head;
    head = temp;
}

void append() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
    temp->data = new_data;
    temp->next = NULL;
    if (head == NULL) {
        head = temp;
        return;
    }
    Node* temp1 = head;
    while (temp1->next != NULL) {
        temp1 = temp1->next;
    }
    temp1->next = temp;
}

void insert() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data, pos;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
    printf("Enter position of the new node: ");
    scanf("%d", &pos);
    temp->data = new_data;
    temp->next = NULL;
    if (pos == 0) {
        temp->next = head;
        head = temp;
        return;
    }
    Node* temp1 = head;
    while (pos--) {
        temp1 = temp1->next;
    }
    Node* temp2 = temp1->next;
    temp->next = temp2;
    temp1->next = temp;
}

void display() {
    Node* temp1 = head;
    while (temp1 != NULL) {
        printf("%d -> ", temp1->data);
        temp1 = temp1->next;
    }
}

```

```

        printf("NULL\n");
    }

void deletePosition()
{
    struct node *temp, *position;
    int i = 1, pos;

    if (start == NULL)
        printf("\nList is empty\n");

    else {
        printf("\nEnter index : ");

        scanf("%d", &pos);
        position = malloc(sizeof(struct node));
        temp = start;

        while (i < pos - 1) {
            temp = temp->link;
            i++;
        }

        position = temp->link;
        temp->link = position->link;

        free(position);
    }
}

```

OUTPUT:

```
C:\Users\Admin\Desktop\1BM22CS192(ds)\Codes\LAB3_prg4.exe
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 0
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 2
Enter data in the new node: 2
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 3
Enter data in the new node: 1
Enter position of the new node: 1
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 4
0 -> 2 -> 1 -> NULL
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 5
Exiting the program
Process returned 0 (0x0)   execution time : 1254.892 s
Press any key to continue.
```

## LEETCODE – 206 Reverse Linked list:

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.

Code:

```
typedef struct {
    int
    stack[300000];
    int min;
    int top;
} MinStack;
```

```
MinStack* minStackCreate() {
    MinStack* obj =
    malloc(sizeof(MinStack)); obj->top
    = -1;
    obj->min = INT_MAX;
    return obj;
}
```

```
void minStackPush(MinStack* obj, int
val) { if (val <= obj->min){
    obj->stack[++(obj->top)] = obj-
    >min; obj->min = val;
}
obj->stack[++(obj->top)] =
val; return;
}
```

```
void minStackPop(MinStack* obj) {
    if (obj->stack[obj->top] == obj->
```

```

        >min){ obj->stack[obj->top] =
        NULL;
        obj->top -= 1;
        obj->min = obj->stack[(obj->top)];
    }
    obj->stack[obj->top] =
    NULL; obj->top -= 1;
}

int minStackTop(MinStack*
obj) { return obj->
stack[obj->top];
}

int minStackGetMin(MinStack* obj)
{ return obj->min;
}

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

```

#### Output:

Accepted a few seconds ago	C	34 ms	18.4 MB
Accepted Jan 11, 2024	C	27 ms	18.5 MB
Accepted Jan 11, 2024	C++	19 ms	16.7 MB

```

1 typedef struct {
2     int stack[300000];
3     int min;
4     int top;
5 } MinStack;
6
7
8 MinStack* minStackCreate() {
9     MinStack* obj = malloc(sizeof(MinStack));
10    obj->top = -1;
11    obj->min = INT_MAX;
12    return obj;
13 }
14
15 void minStackPush(MinStack* obj, int val) {
16     if (val <= obj->min){
17         obj->stack[++(obj->top)] = obj->min;
18         obj->min = val;
19     }
20    obj->stack[++(obj->top)] = val;
21    return;
22 }
23
24 void minStackPop(MinStack* obj) {
25     if (obj->stack[obj->top] == obj->min){
26         obj->stack[obj->top] = NULL;
27         obj->top -= 1;
28         obj->min = obj->stack[(obj->top)];
29     }
30    obj->stack[obj->top] = NULL;

```

## Lab program 5:

### WAP to Implement Singly Linked List with following operations

- a) Create a linked list.
- b) Deletion of first element, specified element and last element in the list.
- c) Display the contents of the linked list.

#### Code:

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

struct node {
    int info;
    struct node* link;
};
struct node* start = NULL;

void createList()
{
    if (start == NULL) {
        int n;
        printf("\nEnter the number of nodes: ");
        scanf("%d", &n);

        if (n > 0) {
            int data;
            struct node* newnode;
            struct node* temp;

            newnode = malloc(sizeof(struct node));
            start = newnode;
            temp = start;

            printf("\nEnter number to be inserted : ");
            scanf("%d", &data);
            start->info = data;

            for (int i = 2; i <= n; i++) {
                newnode = malloc(sizeof(struct node));
                temp->link = newnode;

                printf("\nEnter number to be inserted : ");
                scanf("%d", &data);
                newnode->info = data;
                temp = temp->link;
            }

            temp->link = NULL;
        }
    }
}
```

```

        printf("\nThe list is created\n");
    } else {
        printf("\nThe list is already created\n");
    }
}

void display()
{
    struct node* temp;

    if (start == NULL)
        printf("\nList is empty\n");

    else {
        temp = start;
        while (temp != NULL) {
            printf("Data = %d\n", temp->info);
            temp = temp->link;
        }
    }
}

void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else {
        temp = start;
        start = start->link;
        free(temp);
    }
}

void deleteEnd()
{
    struct node *temp, *prevnode;
    if (start == NULL)
        printf("\nList is Empty\n");
    else {
        temp = start;
        while (temp->link != NULL) {
            prevnode = temp;
            temp = temp->link;
        }
        free(temp);
        prevnode->link = NULL;
    }
}

void deletePosition()
{

```

```

struct node *temp, *position, *prevnode;
int i = 1, pos;

if (start == NULL)
    printf("\nList is empty\n");
else {
    printf("\nEnter index : ");
    scanf("%d", &pos);

    if (pos <= 0) {
        printf("\nInvalid position\n");
        return;
    }

    temp = start;
    position = NULL;

    if (pos == 1) {
        start = start->link;
        free(temp);
        return;
    }

    while (i < pos && temp != NULL) {
        prevnode = temp;
        temp = temp->link;
        i++;
    }

    if (temp == NULL) {
        printf("\nInvalid position\n");
        return;
    }

    position = temp;
    prevnode->link = temp->link;
    free(position);
}

}

int main()
{
    createList();
    int choice;
    while (1) {

        printf("\n\t1. To see list\n");
        printf("\t2. For deletion of "
            "first element\n");
        printf("\t3. For deletion of "
            "last element\n");
        printf("\t4. For deletion of "
            "element at any position\n");
        printf("\t5. To exit\n");
        printf("\nEnter Choice :\n");
        scanf("%d", &choice);
    }
}

```



```
        switch (choice) {
        case 1:
            display();
            break;
        case 2:
            deleteFirst();
            break;
        case 3:
            deleteEnd();
            break;
        case 4:
            deletePosition();
            break;
        case 5:
            exit(1);
            break;
        default:
            printf("Incorrect Choice\n");
        }
    }
    return 0;
}
```

OUTPUT :

```
C:\Users\Admin\Desktop\1BM22CS192(ds)\Codes\LAB4_LLprg5.exe

Enter the number of nodes: 4

Enter number to be inserted : 0

Enter number to be inserted : 1

Enter number to be inserted : 2

Enter number to be inserted : 3

The list is created

    1. To see list
    2. For deletion of first element
    3. For deletion of last element
    4. For deletion of element at any position
    5. To exit

Enter Choice :
2

    1. To see list
    2. For deletion of first element
    3. For deletion of last element
    4. For deletion of element at any position
    5. To exit

Enter Choice :
1
Data = 1
Data = 2
Data = 3

    1. To see list
    2. For deletion of first element
    3. For deletion of last element
    4. For deletion of element at any position
    5. To exit

Enter Choice :
5

Process returned 1 (0x1)   execution time : 34.310 s
Press any key to continue.
```

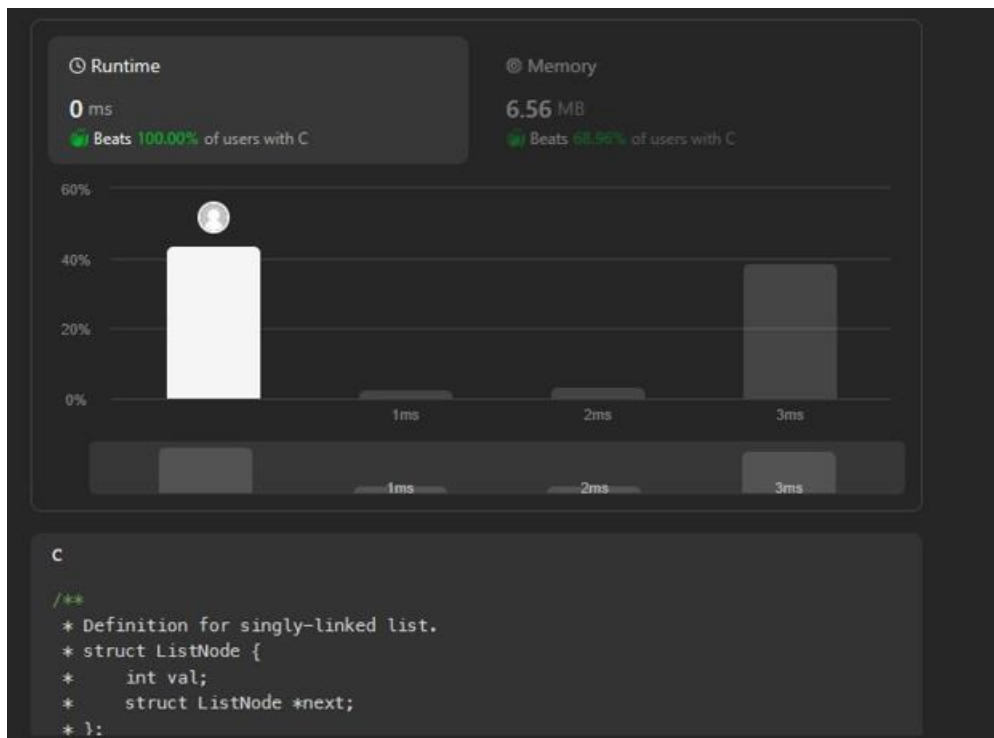
## Program - Leetcode platform

Given the head of a singly linked list, reverse the list, and return *the reversed list*.

### Code:

```
struct ListNode* reverseBetween(struct ListNode* head, int left,
int right) {
    struct ListNode* l =
    head; struct ListNode* r
    = head;
    int difference = right - left;
    if (left == right){
        return head;
    }
    for(int i = 0; i < left-1; i++){
        l = l->next;
    }
    for(int i = 0; i < right-1;
        i++){ r = r->next;
    }
    printf("%d\n%d\n", l->val, r-
>val); while (difference >= 0){
        // printf("%d %d\n", l->val, r-
>val); int temp = l->val;
        l->val = r-
>val; r->val =
        temp;
        l = l-
>next; r =
        l;
        for(int i = 0; i < difference-2;
            i++){ r = r->next;
        }
        difference -=2;
    }
    return head;
}
```

## OUTPUT:



## Lab program 6:

a) WAP to Implement Single Link List with following operations:

- Sort the linked list
- Reverse the linked list
- Concatenation of two linked lists.

### Code:

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node* next;
};
struct node *head, *headB;
void createA(){

    int A[]={2,1,3,5,4,8,6,7,0,9};
    printf("Elements of Linked List A:  ");
    for(int i=0;i<10;i++){
        printf("%d->",A[i]);
    }
    printf("NULL");
    struct node *t, *last;
    head = (struct node*)malloc(sizeof(struct node));
    head->data = A[0];
    head->next = NULL;
    last = head;
    for(int i=1; i<10; i++){
        t = (struct node*)malloc(sizeof(struct node));
        t->data = A[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}

void createB(){

    int B[]={11,13,12,10,14};
    printf("\nElements of Linked List B:  ");
    for(int i=0;i<5;i++){
        printf("%d->",B[i]);
    }
    printf("NULL");
    struct node *t, *last;
    headB = (struct node*)malloc(sizeof(struct node));
    headB->data = B[0];
    headB->next = NULL;
    last = headB;
    for(int i=1; i<5; i++){
        t = (struct node*)malloc(sizeof(struct node));
```

```

        t->data = B[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}

struct node* sort(struct node *head) {
    struct node *ptr1,*ptr2;
    ptr1=head;
    int temp,count=1;
    while(ptr1->next!=NULL) {
        count++;
        ptr1=ptr1->next;
    }
    while(--count){
        ptr1=head;
        while(ptr1->next!=NULL) {
            ptr2=ptr1;
            ptr1=ptr1->next;
            if(ptr1->data<ptr2->data){
                temp=ptr2->data;
                ptr2->data=ptr1->data;
                ptr1->data=temp;
            }
        }
    }
    return head;
}

struct node* concat(struct node *head, struct node *headB){
    struct node *ptr=head;
    while(ptr->next!=NULL) {
        ptr=ptr->next;
    }
    ptr->next=headB;
    return head;
}

struct node* reverse(struct node *head){
    struct node *pre, *cur, *after;
    pre = NULL;
    after = cur = head;
    while(after != NULL)
    {
        after = after->next;
        cur->next = pre;
        pre = cur;
        cur = after;
    }
    head = pre;
    return head;
}

void main(){
    createA();
    createB();
    head = sort(head);
    struct node *p=head;
    printf("\nSorted Linked List A:  ");

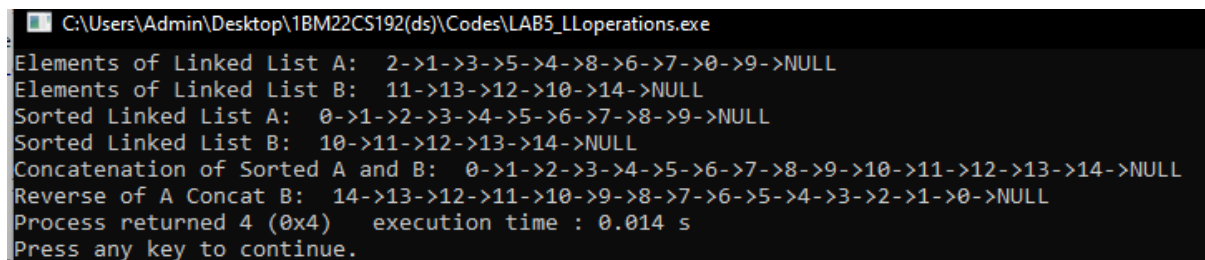
```

```

while(p!=NULL) {
    printf("%d->",p->data);
    p = p->next;
}
printf("NULL");
headB = sort(headB);
p = headB;
printf("\nSorted Linked List B:  ");
while(p!=NULL) {
    printf("%d->",p->data);
    p = p->next;
}
printf("NULL");
p=concat(head,headB);
printf("\nConcatenation of Sorted A and B:  ");
while(p!=NULL) {
    printf("%d->",p->data);
    p = p->next;
}
printf("NULL\n");
p=reverse(head);
printf("Reverse of A Concat B:  ");
while(p!=NULL) {
    printf("%d->",p->data);
    p = p->next;
}
printf("NULL");
}

```

## OUTPUT:



```

C:\Users\Admin\Desktop\1BM22CS192(ds)\Codes\LAB5_LLoperations.exe
Elements of Linked List A:  2->1->3->5->4->8->6->7->0->9->NULL
Elements of Linked List B:  11->13->12->10->14->NULL
Sorted Linked List A:  0->1->2->3->4->5->6->7->8->9->NULL
Sorted Linked List B:  10->11->12->13->14->NULL
Concatenation of Sorted A and B:  0->1->2->3->4->5->6->7->8->9->10->11->12->13->14->NULL
Reverse of A Concat B:  14->13->12->11->10->9->8->7->6->5->4->3->2->1->0->NULL
Process returned 4 (0x4)   execution time : 0.014 s
Press any key to continue.

```

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

### CODE:

#### LL Implementation of Stack:

```

#include<stdio.h>
#include<stdlib.h>
struct Node{
    int val;
    struct Node *next;
}Node;

struct Node *top=NULL;

```

```

void pop() {
    if(top==NULL) {
        printf("Empty stack");
    }
    struct Node *ptr=top;
    top=top->next;
    int num= ptr->val;
    free(ptr);
    printf("Popped element: %d\n",num);
}

void push() {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    int new_data;
    printf("Enter element to add to stack: ");
    scanf("%d", &new_data);
    temp->val = new_data;
    temp->next = top;
    top=temp;
}

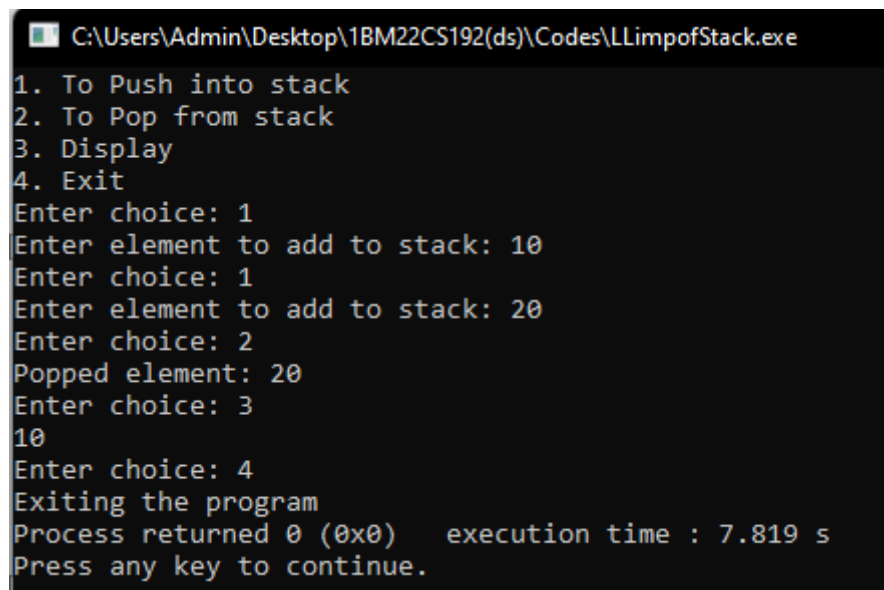
void display() {
    struct Node* temp1 = top;
    while (temp1 != NULL) {
        printf("%d\n", temp1->val);
        temp1 = temp1->next;
    }
}

int main(){
    int choice;
    printf("1. To Push into stack\n");
    printf("2. To Pop from stack\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter choice: ");
    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            default:
                printf("Exiting the program");
                return 0;
        }
        printf("Enter choice: ");
    }
}

```



## OUTPUT:



```
C:\Users\Admin\Desktop\1BM22CS192(ds)\Codes\LLimpofStack.exe
1. To Push into stack
2. To Pop from stack
3. Display
4. Exit
Enter choice: 1
Enter element to add to stack: 10
Enter choice: 1
Enter element to add to stack: 20
Enter choice: 2
Popped element: 20
Enter choice: 3
10
Enter choice: 4
Exiting the program
Process returned 0 (0x0)   execution time : 7.819 s
Press any key to continue.
```

## LL Implementation of Queue:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
    int val;
    struct Node *next;
}Node;

Node *head=NULL;

void enqueue(){
    Node *ptr=(Node*)malloc(sizeof(Node));
    printf("Enter element to enqueue: ");
    int num;
    scanf("%d",&num);
    ptr->val=num;
    if(head==NULL){
        head=ptr;
        ptr->next=NULL;
    }
    else{
        Node *ptr2=head;
        while(ptr2->next!=NULL){
            ptr2=ptr2->next;
        }
        ptr2->next=ptr;
    }
}

void display(){
    Node *ptr=head;
    while(ptr!=NULL){
        printf("%d",ptr->val);
        ptr=ptr->next;
    }
}
```

```

        printf("\n");
    }

void dequeue(){
    Node *ptr=head;
    int num=ptr->val;
    head=head->next;
    free(ptr);
    printf("Dequeued element: %d\n",num);
}

int main(){
    int choice;
    printf("1. To Enqueue into Queue\n");
    printf("2. To Dequeue from Queue\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter choice: ");
    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            default:
                printf("Exiting the program");
                return 0;
        }
        printf("Enter choice: ");
    }
}

```

## OUTPUT:

```
1. Insert
2. Delete
3. Exit
Choice: 1
Enter data: 1
Count: 1
Queue: 1
Enter choice: 1
Enter data: 2
Count: 2
Queue: 1 2
Enter choice: 1
Enter data: 3
Count: 3
Queue: 1 2 3
Enter choice: 2
Integer popped = 1
Count: 2
Queue: 2 3
Enter choice: 2
Integer popped = 2
Count: 1
Queue: 3
Enter choice: 3

Process returned 0 (0x0)   execution time : 8.781 s
Press any key to continue.
```

## Lab 7

### WAP to Implement doubly link list with primitive operations

- a) Create a doubly linked list.
- b) Insert a new node to the left of the node.
- c) Delete the node based on a specific value
- d) Display the contents of the list

#### CODE:

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

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

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

void insertLeft(struct Node** head, int value, int target) {
    struct Node* newNode = createNode(value);
    struct Node* current = *head;

    while (current != NULL && current->data != target) {
        current = current->next;
    }

    if (current != NULL) {
        newNode->prev = current->prev;
        newNode->next = current;
        if (current->prev != NULL) {
            current->prev->next = newNode;
        } else {
            *head = newNode;
        }
        current->prev = newNode;
    } else {
        printf("Node with value %d not found.\n", target);
    }
}

void deleteNode(struct Node** head, int value) {
    struct Node* current = *head;

    while (current != NULL && current->data != value) {
        current = current->next;
    }
}
```

```

    if (current != NULL) {
        if (current->prev != NULL) {
            current->prev->next = current->next;
        } else {
            *head = current->next;
        }

        if (current->next != NULL) {
            current->next->prev = current->prev;
        }

        free(current);
    } else {
        printf("Node with value %d not found.\n", value);
    }
}

void displayList(struct Node* head) {
    struct Node* current = head;

    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }

    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int choice, value, target;

    do {

        printf("\nMenu:\n");
        printf("1. Create a doubly linked list\n");
        printf("2. Insert a new node to the left of the node\n");
        printf("3. Delete the node based on a specific value\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:

                if (head != NULL) {
                    printf("Doubly linked list already exists.\n");
                } else {
                    head = createNode(1);
                    head->next = createNode(2);
                    head->next->prev = head;
                    head->next->next = createNode(3);
                    head->next->next->prev = head->next;
                    printf("Doubly linked list created.\n");
                }
                break;

```

```

e
    case 2:

        printf("Enter the value to be inserted: ");
        scanf("%d", &value);
        printf("Enter the target value: ");
        scanf("%d", &target);
        insertLeft(&head, value, target);
        printf("After insertion: ");
        displayList(head);
        break;

    case 3:

        printf("Enter the value to be deleted: ");
        scanf("%d", &value);
        deleteNode(&head, value);
        printf("After deletion: ");
        displayList(head);
        break;

    case 4:

        printf("Exiting the program.\n");
        break;

    default:
        printf("Invalid choice. Please enter a valid
option.\n");
        }
    } while (choice != 4;

    return 0;
}

```

## OUTPUT:

```
Menu:
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) Exit
Enter your choice: 1
Doubly linked list created.

Menu:
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) Exit
Enter your choice: 2
Enter the value to be inserted: 2
Enter the target value: 3
After insertion: 1 -> 2 -> 2 -> 3 -> NULL

Menu:
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) Exit
Enter your choice: 2
Enter the value to be inserted: 6
Enter the target value: 8
Node with value 8 not found.
After insertion: 1 -> 2 -> 2 -> 3 -> NULL

Menu:
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) Exit
Enter your choice: 2
Enter the value to be inserted: 6
Enter the target value: 3
After insertion: 1 -> 2 -> 2 -> 6 -> 3 -> NULL

Menu:
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) Exit
Enter your choice: 3
Enter the value to be deleted: 2
After deletion: 1 -> 2 -> 6 -> 3 -> NULL

Menu:
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) Exit
Enter your choice:
```

### Leetcode 3:

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.

### CODE:

```
struct ListNode** splitListToParts(struct ListNode* head, int
k,
int* returnSize) {
    struct ListNode* temp = head; int n = 0;
    for(; temp != NULL; temp=temp->next,
        n++); struct ListNode** lists = (struct
ListNode**)malloc(k*sizeof(struct
ListNode*)); for(int i = 0; i < k; i++)
    lists[i] = NULL; int earlier_lists =
n%k, size=n/k;
    int current = 0; bool list_over =
false; temp = head;
    *returnSize = k;
    for(int i = earlier_lists; i > 0; i--){
        // printf("Entering
        here\n"); struct ListNode*
        temp1 = temp;
        lists[current++] = temp;
        for(int j = 0; j < size; j++) temp1 = temp1-
            >next; temp = temp1->next;
        temp1->next = NULL;
    }
    // printf("%d %d %d", lists[0]->val,
    lists[1]->val, lists[2]->val);
    if (temp == NULL) return lists;
    for(int i = 0; i < k-earlier_lists; i++){
        struct ListNode* temp1 = temp;
        if (temp1 == NULL) break;
        for(int j = 0; j < size-1; j++) temp1 = temp1-
            >next; lists[current++] = temp;
        temp = temp1-
            >next; temp1-
            >next = NULL;
        // for(int l = 0; l < k; l++){
```



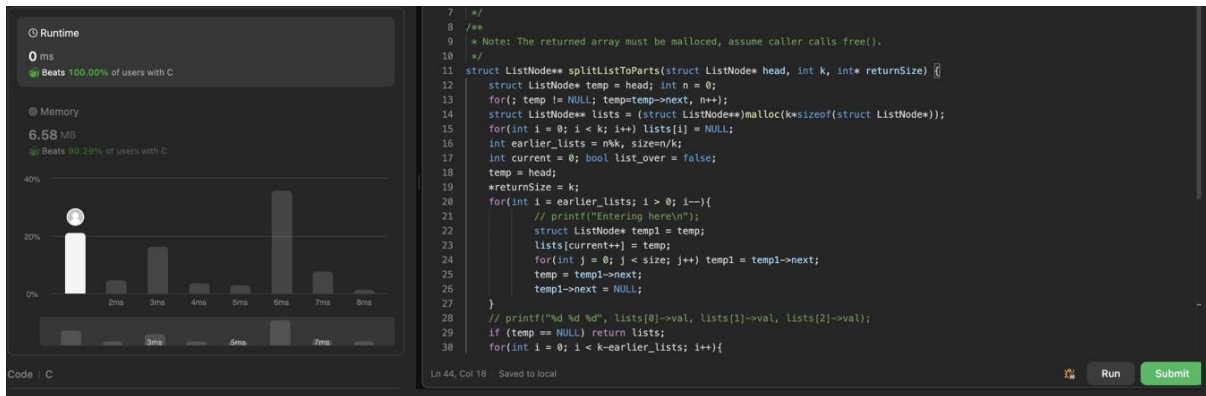
```

        //      for(struct ListNode* temp2 = lists[1];
temp2 != NULL; temp2 = temp2->next){
        //      printf("%d ", temp2->val);
        //      }
        //      printf("\n");
        //  }

    }
    return lists;
}

```

OUTPUT:



## Lab program 8:

Write a program

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

**CODE:**

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

struct node *newNode(int data) {

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

    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return node;
}

struct node *insert(struct node *root, int data) {

    if (root == NULL) {
        return newNode(data);
    }

    else {
        if (data < root->data) {
            root->left = insert(root->left, data);
        }
        else if (data > root->data) {
            root->right = insert(root->right, data);
        }
    }

    return root;
}

void inOrder(struct node *root) {

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

```

void preOrder(struct node *root) {

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

void postOrder(struct node *root) {

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

void display(struct node *root) {

    printf("In-order traversal: ");
    inOrder(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preOrder(root);
    printf("\n");
    printf("Post-order traversal: ");
    postOrder(root);
    printf("\n");
}

int main() {

    struct node *root = NULL;

    int data;

    printf("Enter the elements of the tree (-1 to stop): ");

    while (scanf("%d", &data) && data != -1) {
        root = insert(root, data);
    }

    display(root);

    free(root);

    return 0;
}

```

## OUTPUT:

```
C:\Users\Admin\Desktop\1BM22CS192(ds)\Codes\LAB7_BST.exe
Enter the elements of the tree (-1 to stop): 1
9
0
4
3
5
2
-1
In-order traversal: 0 1 2 3 4 5 9
Pre-order traversal: 1 0 9 4 3 2 5
Post-order traversal: 0 2 3 5 4 9 1

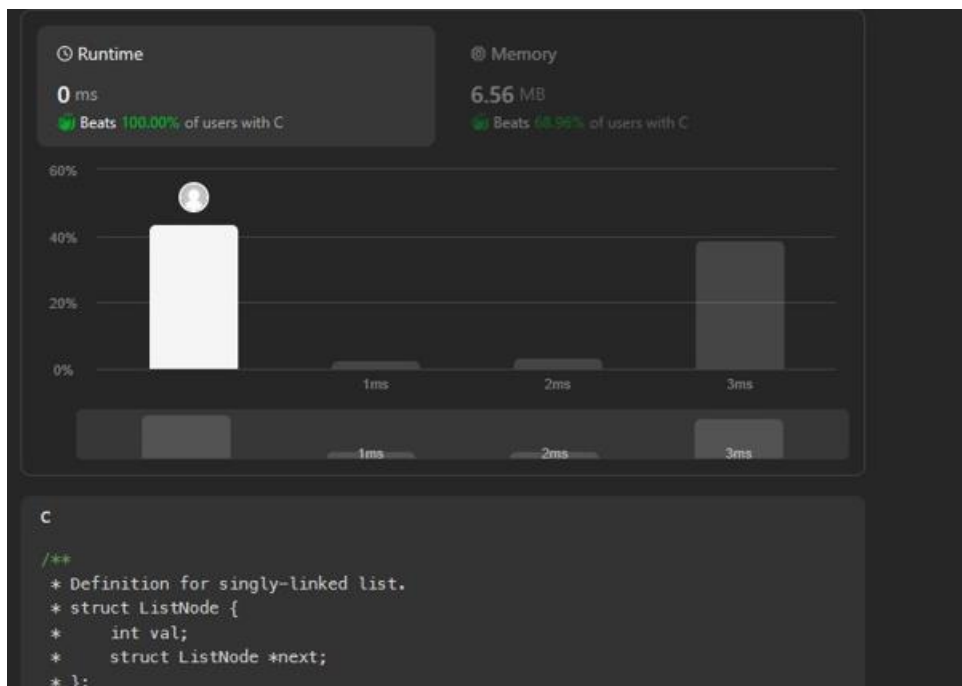
Process returned 0 (0x0)   execution time : 16.434 s
Press any key to continue.
```

## Leetcode 4

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

### Code:

```
struct ListNode* rotateRight(struct ListNode* head, int
    k) { struct ListNode *temp = head;
    if (head == NULL) return NULL;
    if (head->next == NULL) return
head; if (k == 0) return head;
    int size = 1;
    for(; temp->next != NULL; temp=temp->next,
size++); k %= size;
    if (k == 0) return
head; temp->next =
head;
    struct ListNode *temp1 = head;
    for(int i = 0; i < (size-k-1); temp1 = temp1->next,
i++); head = temp1->next;
    temp1->next = NULL;
    return head;
}
```



## Lab program 9:

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

### CODE:

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

#define MAX 100

// Function to perform BFS traversal
void BFS(int adjacency_matrix[MAX][MAX], int visited[MAX], int n, int
start) {
    int queue[MAX], front = -1, rear = -1;

    // Mark the start node as visited and enqueue it
    visited[start] = 1;
    queue[++rear] = start;

    while (front != rear) {
        int current = queue[++front];
        printf("%d ", current + 1);

        // Enqueue adjacent nodes that are not visited
        for (int i = 0; i < n; i++) {
            if (adjacency_matrix[current][i] == 1 && !visited[i]) {
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }
}

int main() {
    int adjacency_matrix[MAX][MAX], visited[MAX], n, i, j;

    // Input number of nodes in the graph
    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    // Input adjacency matrix
    printf("Enter the adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        visited[i] = 0; // Initialize visited array
        for (j = 0; j < n; j++) {
            scanf("%d", &adjacency_matrix[i][j]);
        }
    }

    // Choose the starting node for BFS traversal
    int start_node;
    printf("Enter the starting node for BFS traversal (1 to %d): ", n);
    scanf("%d", &start_node);

    // Perform BFS traversal
    printf("BFS Traversal starting from node %d: ", start_node);
```

```

        BFS(adjacency_matrix, visited, n, start_node - 1);

    return 0;
}

```

## OUTPUT:

```

Enter the number of nodes: 4
Enter the adjacency matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
Enter the starting node for BFS traversal (1 to 4): 2
BFS Traversal starting from node 2: 2 1 3 4
Process returned 0 (0x0)   execution time : 3.602 s
Press any key to continue.

```

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

### CODE:

```

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

#define MAX 100

// Function to perform DFS traversal
void DFS(int adjacency_matrix[MAX][MAX], int visited[MAX], int n, int
current) {
    visited[current] = 1;

    for (int i = 0; i < n; i++) {
        if (adjacency_matrix[current][i] == 1 && !visited[i]) {
            DFS(adjacency_matrix, visited, n, i);
        }
    }
}

// Function to check if the graph is connected using DFS
int isConnected(int adjacency_matrix[MAX][MAX], int visited[MAX], int
n) {
    for (int i = 0; i < n; i++) {
        visited[i] = 0; // Initialize visited array
    }

    DFS(adjacency_matrix, visited, n, 0); // Perform DFS starting from
node 0

    // Check if all nodes are visited
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            return 0; // Graph is not connected
        }
    }
}

```

```

        return 1; // Graph is connected
    }

int main() {
    int adjacency_matrix[MAX][MAX], visited[MAX], n, i, j;

    // Input number of nodes in the graph
    printf("Enter the number of nodes: ");
    scanf("%d", &n);

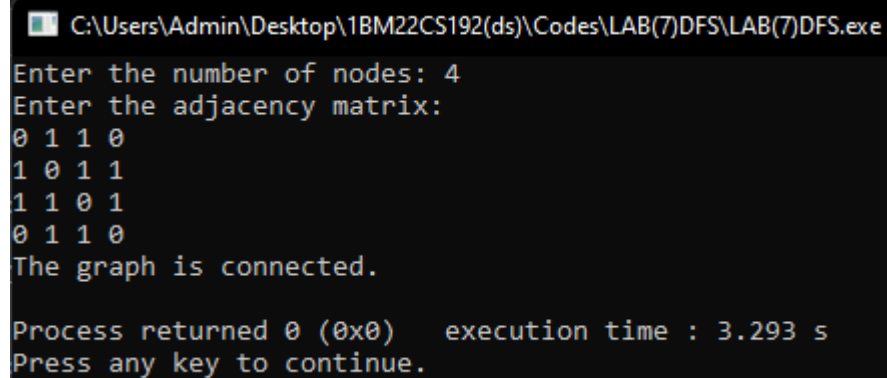
    // Input adjacency matrix
    printf("Enter the adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &adjacency_matrix[i][j]);
        }
    }

    // Check if the graph is connected
    if (isConnected(adjacency_matrix, visited, n)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}

```

### OUTPUT:



```

C:\Users\Admin\Desktop\1BM22CS192(ds)\Codes\LAB(7)DFS\LAB(7)DFS.exe
Enter the number of nodes: 4
Enter the adjacency matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
The graph is connected.

Process returned 0 (0x0)   execution time : 3.293 s
Press any key to continue.

```



### Hackerrank Problem:

Given a tree and an integer,  $k$ , in one operation, we need to swap the subtrees of all the nodes at each depth  $h$ , where  $h \in [k, 2k, 3k, \dots]$ . In other words, if  $h$  is a multiple of  $k$ , swap the left and right subtrees of that level.

You are given a tree of  $n$  nodes where nodes are indexed from  $[1..n]$  and it is rooted at 1. You have to perform  $t$  swap operations on it, and after each swap operation print the in-order traversal of the current state of the tree.

### CODE:

```
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

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

void inOrderTraversal(Node* root, int* result, int* index) {
    if (root == NULL) return;
    inOrderTraversal(root->left, result, index);
    result[(*index)++] = root->data;
    inOrderTraversal(root->right, result, index);
}

void swapAtLevel(Node* root, int k, int level) {
    if (root == NULL) return;
    if (level % k == 0) {
        Node* temp = root->left;
        root->left = root->right;
        root->right = temp;
    }
    swapAtLevel(root->left, k, level + 1);
    swapAtLevel(root->right, k, level + 1);
}
```

```

int** swapNodes(int indexes_rows, int indexes_columns, int** indexes,
int queries_count, int* queries, int* result_rows, int* result_columns)
{
    // Build the tree
    Node** nodes = (Node**)malloc((indexes_rows + 1) * sizeof(Node*));
    for (int i = 1; i <= indexes_rows; i++) {
        nodes[i] = createNode(i);
    }

    for (int i = 0; i < indexes_rows; i++) {
        int leftIndex = indexes[i][0];
        int rightIndex = indexes[i][1];
        if (leftIndex != -1) nodes[i + 1]->left = nodes[leftIndex];
        if (rightIndex != -1) nodes[i + 1]->right = nodes[rightIndex];
    }

    // Perform swaps and store results
    int** result = (int**)malloc(queries_count * sizeof(int*));
    *result_rows = queries_count;
    *result_columns = indexes_rows;
    for (int i = 0; i < queries_count; i++) {
        swapAtLevel(nodes[1], queries[i], 1);
        int* traversalResult = (int*)malloc(indexes_rows *
sizeof(int));
        int index = 0;
        inOrderTraversal(nodes[1], traversalResult, &index);
        result[i] = traversalResult;
    }

    free(nodes);
    return result;
}

int main() {
    int n;
    scanf("%d", &n);

    int** indexes = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        indexes[i] = malloc(2 * sizeof(int));
        scanf("%d %d", &indexes[i][0], &indexes[i][1]);
    }

    int queries_count;
    scanf("%d", &queries_count);

    int* queries = malloc(queries_count * sizeof(int));
    for (int i = 0; i < queries_count; i++) {
        scanf("%d", &queries[i]);
    }

    int result_rows;
    int result_columns;
    int** result = swapNodes(n, 2, indexes, queries_count, queries,
&result_rows, &result_columns);

    for (int i = 0; i < result_rows; i++) {
        for (int j = 0; j < result_columns; j++) {

```

```

        printf("%d ", result[i][j]);
    }
    printf("\n");
    free(result[i]); // Free memory allocated for each row
}
free(result); // Free memory allocated for the result array

// Free memory allocated for indexes and queries arrays
for (int i = 0; i < n; i++) {
    free(indexes[i]);
}
free(indexes);
free(queries);

return 0;
}

```

OUTPUT:

## 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)

1	3
2	2 3
3	-1 -1
4	-1 -1
5	2
6	1
7	1

Your Output (stdout)

1	3 1 2
2	2 1 3