

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

NIHAL REDDY S (1BM22CS179)

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 **NIHAL REDDY S (1BM22CS179)**, 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

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	Infix to Postfix expression	6
3	Queue & Circular Queue Implementation	9
4	Singly Linked List & Leetcode-min stack	14
5	Singly Linked List(Deletion) & Leetcode- Reverse list	21
6	Linked List Operations,Stack & Queue implementation Using Linked List	28
7	Doubly Linked List & Leetcode – split linked list	43
8	Binary Search Tree & Leetcode- Rotate List	52
9	Traverse Graph using BFS method	59
10	Hashing Program	63

Course outcomes:

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

Lab program 1:

Write a program to simulate the working of stack using an array with the following:

- a) Push
- b) Pop
- c) Display

The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>
#include <stdlib.h>
#define STACK_SIZE 5
void push(int st[],int *top)
{
    int item;
    if(*top==STACK_SIZE-1)
        printf("Stack overflow\n");
    else
    {
        printf("\nEnter an item :");
        scanf("%d",&item); (*top)
        ++;
        st[*top]=item;
    }
}
void pop(int st[],int *top)
{
    if(*top== -1)
        printf("Stack underflow\n");
    else
    {
        printf("\n%d item was deleted",st[(*top)--]);
    }
}
void display(int st[],int *top)
{
    int i;
    if(*top== -1)
        printf("Stack is empty\n");
    for(i=0;i<=*top;i++)
        printf("%d\t",st[i]);
}
void main()
{
    int st[10],top=-1, c,val_del;
    while(1)
    {
        printf("\n1. Push\n2. Pop\n3. Display\n");
        printf("\nEnter your choice :");
        scanf("%d",&c);
```

```

switch(c)
{
    case 1: push(st,&top);
            break;
    case 2: pop(st,&top);
            break;
    case 3: display(st,&top);
            break;
    default: printf("\nInvalid choice!!!");
            exit(0);
}
}
}

```

Output:

```

D:\II SEM\New folder\2.exe
Enter the option for the following operations:
1.Push
2.Pop
3.Display
4.Exit
1
Enter the element to be added
25
Enter the option for the following operations:
1.Push
2.Pop
3.Display
4.Exit
2
Popped element is 25
Enter the option for the following operations:
1.Push
2.Pop
3.Display
4.Exit
3
Stack is empty
Enter the option for the following operations:
1.Push
2.Pop
3.Display
4.Exit
4
Process returned 0 (0x0)   execution time : 14.555 s

```

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>
#include<ctype.h>
#define max 20
void push(char a);
char pop();
char stack[max],top=-1;
int pre(char a);

void main(){
    char infix[max],a;
    char post[max];
    printf("Enter infix expression: ");
    scanf("%s",infix);
    int j=0;
    push('(');
    for(int i=0;i<strlen(infix);i++){
        if(isalnum(infix[i]))
            { post[j]=infix[i];
              j+=1;
            }
        else if((infix[i]=='+' || infix[i]=='-' || infix[i]=='/' || infix[i]=='*')){
            if(pre(infix[i])>pre(stack[top])){
                push(infix[i]);
            }
            else if(pre(infix[i])<=pre(stack[top])){

                while(1)
                { a=pop()
                  ; if(a=='(')
                  {
                      push(a);
                      break;
                  }
                  post[j]=a;
                  j+=1;
                }
                push(infix[i]);
            }
        }
    }
    while(top!=-1)
    { char
      y=pop();
```

```

        break;
    }
    post[j]=y;
    j+=1;

}
post[j]='\0';
printf("%s",post);

}

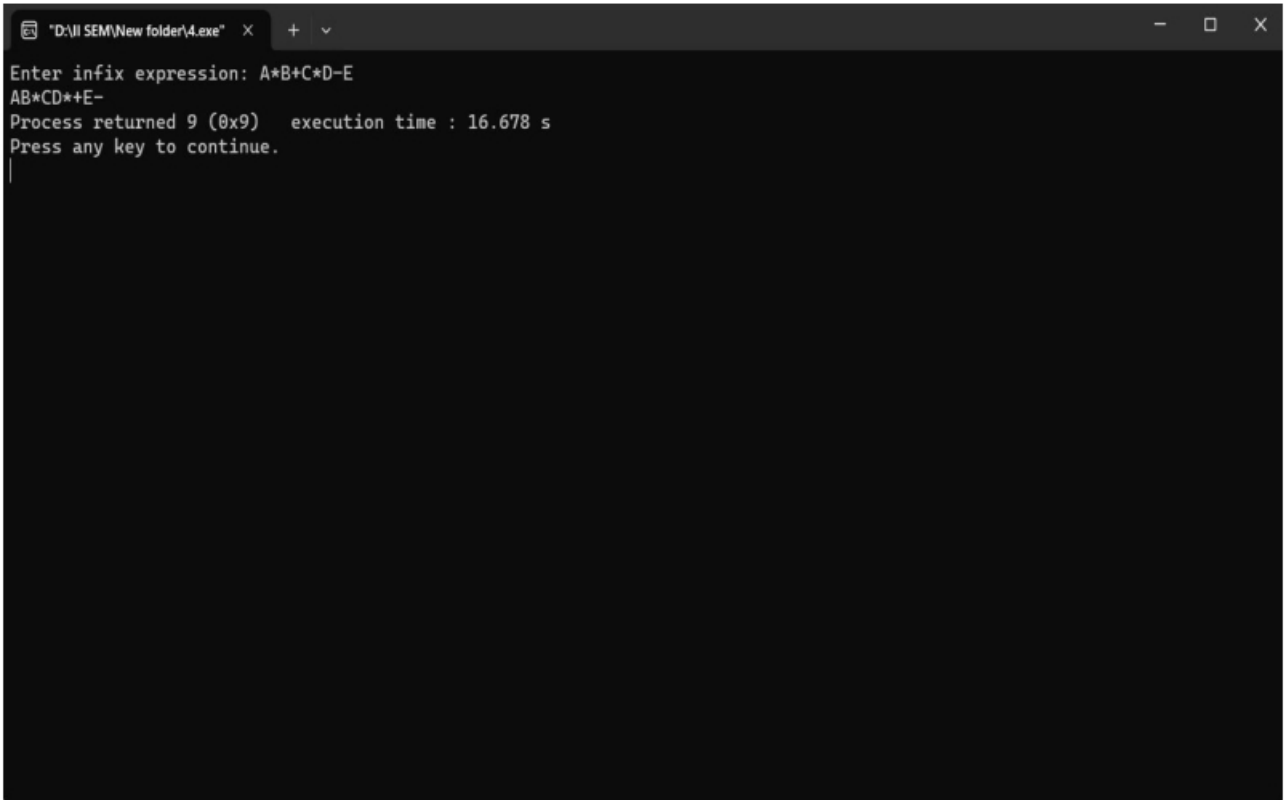
void push(char a){
    if(top>max-1)
        { printf("Stack
        overflow"); exit(0);
        }
    else{
        ++top;
        stack[top]=a;
    }
}

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

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

```

Output:



```
"D:\II SEM\New folder\4.exe" x + v
Enter infix expression: A*B+C*D-E
AB*CD**E-
Process returned 9 (0x9)   execution time : 16.678 s
Press any key to continue.
|
```


Lab Program 3:

3a) WAP to simulate the working of a queue of integers using an array.

Provide the following operations: Insert, Delete, Display

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

```
#include<stdio.h>
#include<string.h>
#define MAX 100
int rear=-1,front=-1;
int queue[MAX];
void enqueue();
int dequeue();
void display();
int main()
{
    int option,val;
    do
    {
        printf("\nEnter an option to perform following
operations\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        scanf("%d",&option);
        switch(option)
        {
            case 1: enqueue();
            break;
            case 2: val=dequeue();
                printf("Element deleted from queue is: %d",val);
            break;
            case 3: display();
            break;
        } }while(option!
=4); return 0;
}
void enqueue()
{
    int x;
    printf("Enter the number to be inserted in the queue\n");
    scanf("%d",&x);
    if(rear==MAX-1)
        printf("Overflow");
    else if(front==-1&&rear== -1)
        front=rear=0;
    else
        rear=rear+1;
    queue[rear]=x;
}
int dequeue()
{

```

```

int y; if(front==-1||
front>rear)
printf("Underflow");
else
{
y=queue[front];
front=front+1;
}
return y;
}
void display()
{
int i;
printf("Elements in the queue:\n");
if(front==-1||front>rear)
printf("Underflow");
for(i=front;i<=rear;i++)
printf("\t%d",queue[i]);
}

```

Output:

```

"D:\II SEM\New folder\6.exe" x + v
Enter an option to perform following operations
1.Insert
2.Delete
3.Display
4.Exit
1
Enter the number to be inserted in the queue
18
Enter an option to perform following operations
1.Insert
2.Delete
3.Display
4.Exit
2
Element deleted from queue is: 18
Enter an option to perform following operations
1.Insert
2.Delete
3.Display
4.Exit
3
Elements in the queue:
Underflow
Enter an option to perform following operations
1.Insert
2.Delete
3.Display
4.Exit
4
Process returned 0 (0x0) execution time : 7.040 s
Press any key to continue.

```

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 MAX 5
int rear = -1, front = -1;
int queue[MAX];

void enqueue();
int dequeue();
void display();

int main() {
    int option, value;
    do {
        printf("\nEnter an option to perform the following operations:\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        scanf("%d", &option);

        switch (option) {
            case 1:
                enqueue();
                break;
            case 2:
                value = dequeue();
                printf("Element deleted from queue: %d\n", value);
                break;
            case 3:
                display();
                break;
        }
    } while (option != 4);

    return 0;
}

void enqueue() {
    int x;
    printf("Enter the element to be inserted: ");
    scanf("%d", &x);

    if ((rear + 1) % MAX == front) {
        printf("Overflow. Queue is full.\n");
    } else if (front == -1 && rear == -1) {
        front = rear = 0;
        queue[rear] = x;
    } else if (front != 0 && rear == MAX - 1) {
        rear = 0;
    }
}
```

```

        queue[rear] = x;
    } else {
        rear = (rear + 1) % MAX;
        queue[rear] = x;
    }
}

int dequeue() {
    int y;
    if (front == -1 && rear == -1)
        { printf("Underflow. Queue is empty.
        \n"); return -1;
        }

    y = queue[front];
    if (front == rear) {
        front = rear = -1;
    } else {
        if (front == MAX - 1) {
            front = 0;
        } else {
            front = (front + 1) % MAX;
        }
    }

    return y;
}

void display() {
    int i;
    printf("Elements in the Queue:\n");

    if (front == -1 && rear == -1) {
        printf("Underflow. Queue is empty.
        \n"); } else {
        if (front <= rear) {
            for (i = front; i <= rear; i++) {
                printf("\t%d", queue[i]);
            }
        } else {
            for (i = front; i < MAX; i++) {
                printf("\t%d", queue[i]);
            }
            for (i = 0; i <= rear; i++) {
                printf("\t%d", queue[i]);
            }
        }
        printf("\n");
    }
}

```

Output:

```
C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Lab3_circular_queue_implementation.exe

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

```
C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\Lab3_circular_queue_implementation.exe

3. Display
4. Exit
1
Enter the element to be inserted into the queue
4
Element inserted!
1. Insert
2. Delete
3. Display
4. Exit
3
Elements of the queue:
2
3
4
1. Insert
2. Delete
3. Display
4. Exit
2
Element 2 is deleted from the queue
1. Insert
2. Delete
3. Display
4. Exit
2
Element 3 is deleted from the queue
1. Insert
2. Delete
3. Display
4. Exit
2
Element 4 is deleted from the queue
1. Insert
2. Delete
3. Display
4. Exit
2
Queue is empty
Process returned 0 (0x0)   execution time : 40.415 s
Press any key to continue.
```

Lab Program 4:

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;
};

struct node *start=NULL;

void insert_begin();
void insert_end();
void insert_pos();
void display();

int main()
{
    int option;
    do{
        printf("\n***MAIN MENU***\n1.Insert at beginning\n2.Insert at end\n3.Insert at any
position\n4.Display\n5.Exit\n");

        printf("\nEnter an option to perform the following operations: ");
        scanf("%d",&option);
        switch(option)
        {
            case 1:insert_begin();
                printf("\nElement inserted successfully\n");
```

```

        break;
    case 2:insert_end();
        printf("\nElement inserted successfully\n");
        break;
    case 3:insert_pos();
        printf("\nElement inserted successfully\n");
        break;
    case 4:printf("\nElements in the linked list:\n");
        display();
        break;
} }while(option!
=5); return 0;

```

```

}
void insert_begin()
{
    struct node *new_node;
    int num;
    printf("Enter the data\n");
    scanf("%d",&num);
    new_node=(struct node*)malloc(sizeof(struct node));
    new_node->data=num;
    new_node->next=start;
    start=new_node;
}
void insert_end()
{
    struct node *new_node,*ptr;
    int num;

15| P a g

```

```

printf("Enter the data\n");
scanf("%d",&num);
new_node=(struct node*)malloc(sizeof(struct node));
new_node->data=num;
new_node->next=NULL;
ptr=start; while(ptr-
>next!=NULL) ptr=ptr-
>next; ptr-
>next=new_node;
}
void insert_pos()
{
    struct node *new_node,*ptr,*prev;
    int num,pos,count=1;
    printf("Enter the data\n");
    scanf("%d",&num);
    printf("Enter the position to be inserted\n");
    scanf("%d",&pos);
    new_node=(struct node*)malloc(sizeof(struct node));
    new_node->data=num;
    if(pos==1)
    { new_node-
    >next=start;
    start=new_node;
    }
    else
    {
        ptr=start;
        while(count<pos&&ptr!=NULL)

```



```

    {
        prev=ptr;
        ptr=ptr->next;
        count++;
    }
    if(count==pos)
    {
        prev->next=new_node;
        new_node->next=ptr;
    }
}
void display()
{
    struct node *ptr;
    ptr=start;
    while(ptr!=NULL)
    {
        printf("\t%d",ptr->data);
        ptr=ptr->next;
    }
    printf("\n");
}

```

Output:

```
"C:\Users\Hp\Desktop\SEM 3" x + v
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: 18
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: 78
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 4
18 -> 78 -> 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 : 11.568 s
Press any key to continue.
```

Leetcode Program: Min Stack

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

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

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

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

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

18| P a g e

```

        obj->min[obj->top] = obj->min[obj->top - 1];
    else
        obj->min[obj->top] = val;
    }
    else
        obj->min[obj->top] = val;
}

```

```

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

```

```

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

```

```

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

```

```

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

```

Output:

The screenshot displays the LeetCode interface for the 'Min Stack' problem. The left panel shows the 'Submissions' tab with a bar chart indicating the submission's performance relative to other users. The right panel shows the 'Code' tab with the C code solution and the 'Test Result' tab showing the test case details.

Runtime: 25 ms (Beats 74.20% of users with C)
Memory: 13.61 MB (Beats 97.97% of users with C)

Code:

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

typedef struct {
    int top;
```

Test Result: Accepted (Runtime: 5 ms)

Case 1:

Input: ["MinStack", "push", "push", "push", "getMin", "pop", "top", "getMin"]

Output: [null, null, null, null, -3, null, 0, -2]

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.**

```
#include<stdio.h>

#include<stdlib.h>

struct node

{
    int data;
    struct node *next;
};

struct node *start=NULL;

void create();

void delete_begin();

void delete_end();

void delete_pos();

void display();

int main()

{
    int option;
    do{
        printf("\n***MAIN MENU***\n1.Create linked list\n2.Delete from beginning\n3.Delete
from end\n4.Delete from any position\n5.Display linked list\n6.Exit\n");

        printf("\nEnter an option to perform the following operations: ");

        scanf("%d",&option);
        switch(option)
        {
            21| P a g e
```

```

        case 1:create();

            printf("\nLinked list created successfully\n");

            break;

        case 2:delete_begin();

            printf("Element deleted successfully\n");

            break;

        case 3:delete_end();

            printf("Element deleted successfully\n");

            break;

        case 4:delete_pos();

            printf("Element deleted successfully\n");

            break;

        case 5:printf("\nElements in the linked list:\n");

            display();

            break;

    } }while(option!

=6); return 0;

}

void create()

{

    struct node *ptr,*new_node;

    int num;

    printf("Enter -1 to exit\n");

    printf("\nEnter the data\n");

    scanf("%d",&num);

    while(num!=-1)

    {

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

```

```

new_node->data=num;
if(start==NULL)
{
    start=new_node;
    new_node->next=NULL;
}
else
{
    ptr=start; while(ptr-
    >next!=NULL) ptr=ptr-
    >next; ptr-
    >next=new_node;
    new_node->next=NULL;
}
printf("Enter the data\n");
scanf("%d",&num);
}
}
void delete_begin()
{
    struct node *ptr;
    ptr=start;
    start=start->next;
    free(ptr);
}
void delete_end()
{
    struct node *ptr,*preptr;
    ptr=start;

```

```

while(ptr->next!=NULL)
{
preptr=ptr;
ptr=ptr-
>next; }

preptr->next=NULL;

free(ptr);
}

void delete_pos()
{
    struct node *ptr,*preptr,*postptr;
    int pos,count=1;
    printf("Enter the position: ");
    scanf("%d",&pos);
    ptr=start;
    if(pos==1)
    {
        start=start->next;
        free(ptr);
    }
    else
    {
        while(count<pos&&ptr!=NULL)
        {
            preptr=ptr;
            ptr=ptr->next;
            postptr=ptr->next;
            count++;
        }

```



```

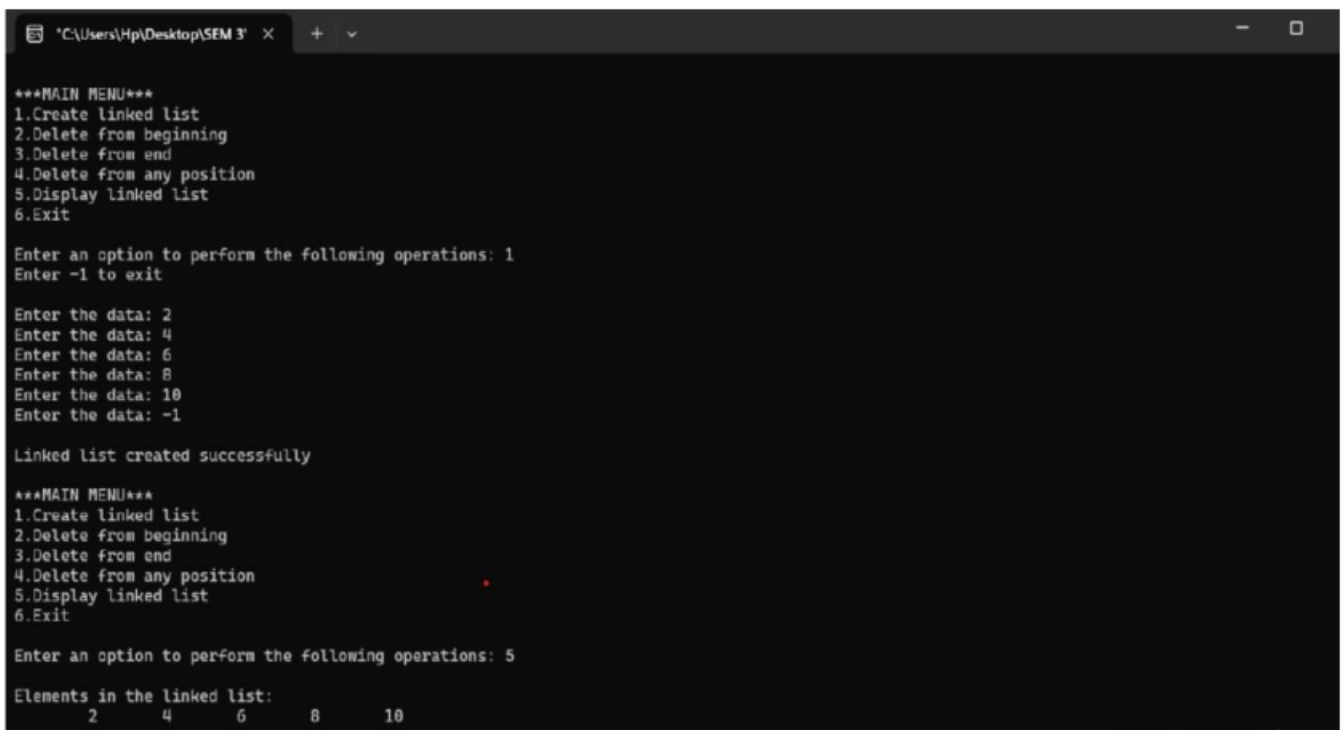
        if(pos==count)
        {
            preptr->next=postptr;
            free(ptr);
        }

    }
}

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

```

Output:



```
***MAIN MENU***
1.Create linked list
2.Delete from beginning
3.Delete from end
4.Delete from any position
5.Display linked list
6.Exit

Enter an option to perform the following operations: 1
Enter -1 to exit

Enter the data: 2
Enter the data: 4
Enter the data: 6
Enter the data: 8
Enter the data: 10
Enter the data: -1

Linked list created successfully

***MAIN MENU***
1.Create linked list
2.Delete from beginning
3.Delete from end
4.Delete from any position
5.Display linked list
6.Exit

Enter an option to perform the following operations: 5

Elements in the linked list:
2 4 6 8 10
```

Leetcode Program: Reverse Linked List

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    struct ListNode* ptrl= head;

    int temp=left-1;

    while(temp--){

        ptrl=ptrl->next;
    }

    int count=right-left+1;

    int* a = (int*)malloc(count * sizeof(int));

    for(int i=0;i<count;i++){

        a[i]=ptrl->val;

        ptrl=ptrl->next;
    }

    struct ListNode* ptr= head;

    left--;

    while(left--){
```

```

printf("%d",ptr->val);

ptr=ptr->next;

}

for(int i=count-1;i>-1;i--){

    ptr->val=a[i];

    ptr=ptr->next;

}

return head;

}

```

Output:

The screenshot displays a C++ IDE with the following components:

- Problem List:** Shows the current problem status as "Accepted".
- Performance Metrics:**
 - Runtime: 0 ms
 - Memory: 6.78 MB
 - Beats: 100.00% of users with C
- Code Editor:** Contains the following C++ code:

```

1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     struct ListNode *next;
6  * };
7  */
8 struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
9     struct ListNode* ptr = head;
10    int temp = left - 1;
11    while(temp--){

```
- Test Result Panel:**
 - Status: Accepted
 - Runtime: 0 ms
 - Case 1: [1,2,3,4,5]
 - left: 2
 - right: 4

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;

};

struct node *s1=NULL;

struct node *s2=NULL;

struct node *start=NULL;

struct node *create(struct node*);

void sort();

struct node *concatenate(struct node*,struct node*);

void reverse();

void display(struct node*);

int main()

{

    int option;

    struct node *a=NULL;

    do{

        printf("\n*****MAIN MENU*****\n\n1.Create a linked list\n2.Create two linked lists for concatenation\n3.Sort\n4.Concatenate\n5.Reverse\n6.Display linked list\n7.Display Concatenated linked list\n8.Exit\n");

        printf("\nEnter an option to perform the following operations: ");

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

```

switch(option)
{
    case 1:start=create(start);
        printf("\nLinked list created successfully\n");
        break;
    case 2:printf("\nLinked list 1:\n");
        s1=create(s1);
        printf("\nLinked list 2:\n");
        s2=create(s2);
        printf("\nLinked lists created successfully\n");
        break;
    case 3:sort();
        printf("\nLinked list sorted\n");
        break;
    case 4:a=concatenate(s1,s2);
        printf("\nLinked lists concatenated successfully\n");
        break;
    case 5:reverse();
        printf("\nLinked list reversed\n");
        break;
    case 6:printf("\nElements in the linked list\n");
        display(start);
        break;
    case 7:printf("\nElements in the linked list after concatenation:\n");
        display(a);
        break;

}
}while(option!=8);

```

```

    return 0;
}

struct node * create(struct node *start)
{
    struct node *ptr,*new_node;
    int num;
    printf("Enter -1 to exit\n");
    printf("\nEnter the data: ");
    scanf("%d",&num);
    while(num!=-1)
    {
        new_node=(struct node*)malloc(sizeof(struct node));
        new_node->data=num;
        if(start==NULL)
        {
            start=new_node;
            new_node->next=NULL;
        }
        else
        {
            ptr=start; while(ptr->next!=NULL) ptr=ptr->next; ptr->next=new_node;
            new_node->next=NULL;
        }
        printf("Enter the data: ");
        scanf("%d",&num);
    }
}

```

```

    }

    return start;
}

void sort()
{
    struct node *i,*j;
    int temp; for(i=start;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;
            }
        }
    }
}

```

```

struct node *concatenate(struct node *t1,struct node *t2)
{
    struct node *ptr;
    ptr=t1;
    while(ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
}

```

```

    } ptr-
>next=t2;
return t1;
}

```

```

void reverse()
{
    struct node *prev=NULL;
    struct node *next=NULL;
    struct node *cur=start;
    while(cur!=NULL)
    {
        next=cur->next;
        cur->next=prev;
        prev=cur;
        cur=next;
    }
    start=prev;
}

```

```

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

```



```
printf("\n");  
}
```

Output:

```
*****MAIN MENU*****  
  
1.Create a linked list  
2.Create two linked lists for concatenation  
3.Sort  
4.Concatenate  
5.Reverse  
6.Display linked list  
7.Display Concatenated linked list  
8.Exit  
  
Enter an option to perform the following operations: 1  
Enter -1 to exit  
  
Enter the data: 2  
Enter the data: 4  
Enter the data: 6  
Enter the data: 8  
Enter the data: 10  
Enter the data: -1  
  
Linked list created successfully  
  
*****MAIN MENU*****  
  
1.Create a linked list  
2.Create two linked lists for concatenation  
3.Sort  
4.Concatenate  
5.Reverse  
6.Display linked list  
7.Display Concatenated linked list  
8.Exit
```

```
6.Display linked list
7.Display Concatenated linked list
8.Exit
```

Enter an option to perform the following operations: 6

Elements in the linked list

10 8 6 4 2

*****MAIN MENU*****

```
1.Create a linked list
2.Create two linked lists for concatenation
3.Sort
4.Concatenate
5.Reverse
6.Display linked list
7.Display Concatenated linked list
8.Exit
```

Enter an option to perform the following operations: 3

Linked list sorted

*****MAIN MENU*****

```
1.Create a linked list
2.Create two linked lists for concatenation
3.Sort
4.Concatenate
5.Reverse
6.Display linked list
7.Display Concatenated linked list
8.Exit
```

Enter an option to perform the following operations: 2

```

Linked list 1:
Enter -1 to exit

Enter the data: 18
Enter the data: 78
Enter the data: 64
Enter the data: -1

Linked list 2:
Enter -1 to exit

Enter the data: 24
Enter the data: 84
Enter the data: -1

Linked lists created successfully

****MAIN MENU****

1.Create a linked list
2.Create two linked lists for concatenation
3.Sort
4.Concatenate
5.Reverse
6.Display linked list
7.Display Concatenated linked list
8.Exit

Enter an option to perform the following operations: 7

Elements in the linked list after concatenation:

****MAIN MENU****

1.Create a linked list
2.Create two linked lists for concatenation
3.Sort
4.Concatenate

```

```

Linked lists concatenated successfully

****MAIN MENU****

1.Create a linked list
2.Create two linked lists for concatenation
3.Sort
4.Concatenate
5.Reverse
6.Display linked list
7.Display Concatenated linked list
8.Exit

Enter an option to perform the following operations: 7

Elements in the linked list after concatenation:
    18     78     64     24     84

****MAIN MENU****

1.Create a linked list
2.Create two linked lists for concatenation
3.Sort
4.Concatenate
5.Reverse
6.Display linked list
7.Display Concatenated linked list
8.Exit

Enter an option to perform the following operations: 8

```

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

Operations. //

Stack Implementation

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

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

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node *next;
```

```
};
```

```
struct node *start=NULL;
```

```
void push();
```

```
void pop();
```

```
void display();
```

```
int main()
```

```
{
```

```
    int val,option;
```

```
    do
```

```
    {
```

```
        printf("\nEnter the number to perform following  
operations\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
```

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

```
        switch(option)
```

```
        {
```

```
            case 1:push();
```

```
            break;
```

```
            case 2:pop();
```

```
            break;
```

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

```
            break;
```

```

    }
} while(option!
=4); return 0;

}

void push()
{
    struct node *new_node;
    int num;
    printf("Enter the data\n");
    scanf("%d",&num);
    new_node=(struct node*)malloc(sizeof(struct node));
    new_node->data=num;
    new_node->next=start;
    start=new_node;
}

void pop()
{
    struct node *ptr;
    ptr=start;
    if(start==NULL)
    {
        printf("Stack is empty\n");
        exit(0);
    }
    else
    {
        ptr=start;
        start=ptr->next;
    }
}

```

```

printf("\nElement popped from the stack is: %d\n",ptr->data);
free(ptr);
}

}

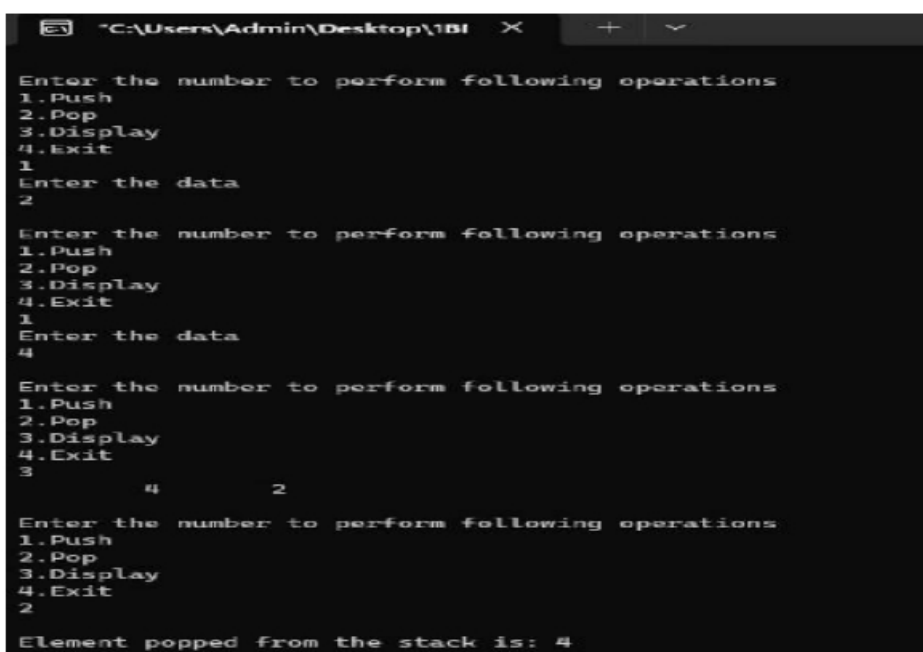
void display()
{

    struct node *ptr;

    ptr=start;
    while(ptr!=NULL)
    {
        printf("\t%d",ptr->data);
        ptr=ptr->next;
    }
    printf("\n");
}

```

Output:



```

C:\Users\Admin\Desktop\1BI X
Enter the number to perform following operations
1.Push
2.Pop
3.Display
4.Exit
1
Enter the data
2
Enter the number to perform following operations
1.Push
2.Pop
3.Display
4.Exit
1
Enter the data
4
Enter the number to perform following operations
1.Push
2.Pop
3.Display
4.Exit
3
4 2
Enter the number to perform following operations
1.Push
2.Pop
3.Display
4.Exit
2
Element popped from the stack is: 4

```

//Queue Implementation

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

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

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node
```

```
*next; };
```

```
struct node *start=NULL;
```

```
void enqueue();
```

```
void dequeue();
```

```
void display();
```

```
int main()
```

```
{
```

```
    int val,option;
```

```
    do
```

```
    {
```

```
        printf("\nEnter the number to perform following  
operations\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
```

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

```
        switch(option)
```

```
        {
```

```
            case 1:enqueue();
```

```
            break;
```

```
            case 2:dequeue();
```

```
            break;
```

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

```
            break;
```

```
        }
```

```

    } while(option!
    =4); return 0;

}

void enqueue()
{
    struct node *new_node;
    int num;
    printf("Enter the data\n");
    scanf("%d",&num);
    new_node=(struct node*)malloc(sizeof(struct node));
    new_node->data=num;
    new_node->next=start;
    start=new_node;
}

void dequeue()
{
    struct node *ptr,*preptr;
    ptr=start;
    if(start==NULL)
    {
        printf("Stack is empty\n");
        exit(0);
    }
    else if(start->next==NULL)
    {
        start=start->next;
        printf("\nElement popped from the stack is: %d\n",ptr->data);
        free(ptr);
    }
}

```



```

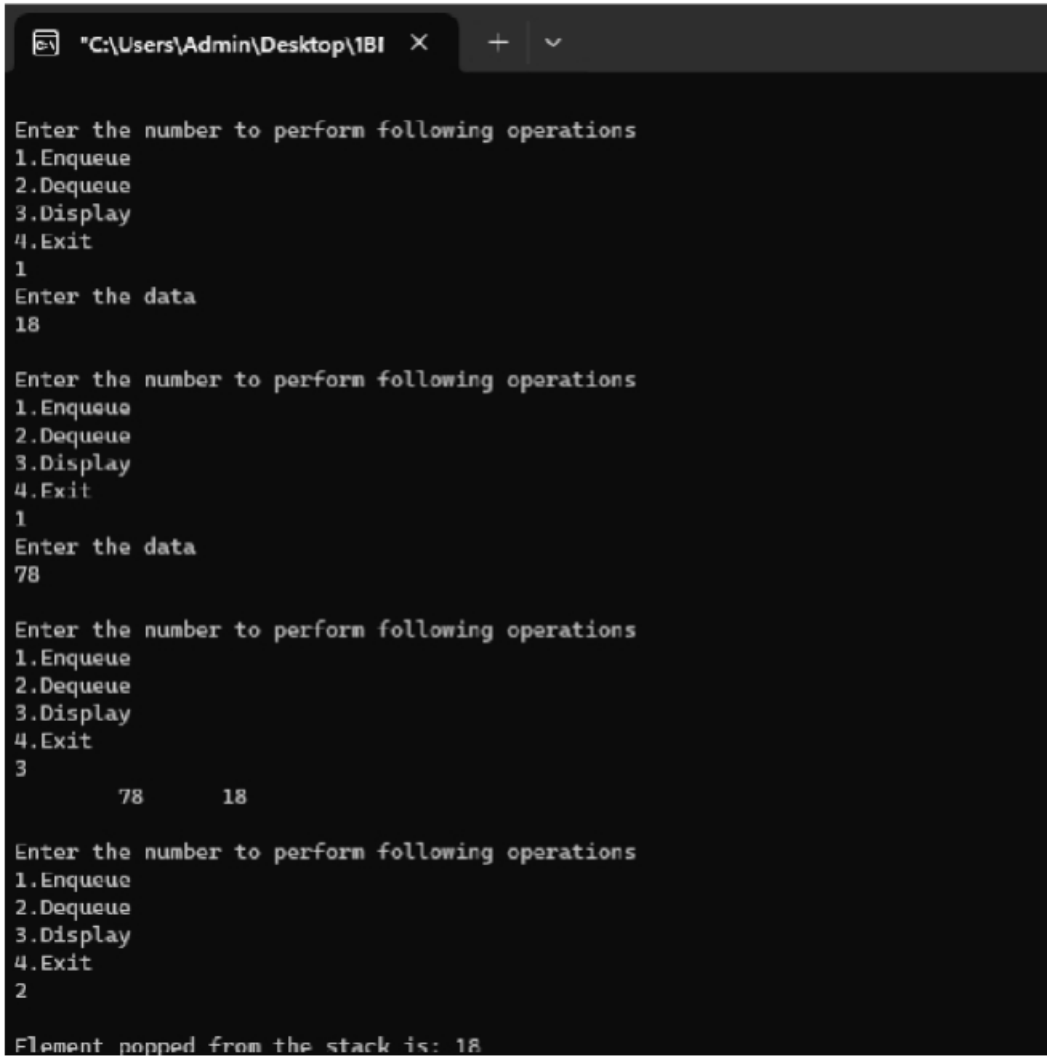
    }
else
{
    while(ptr->next!=NULL)
    {
        preptr=ptr;
        ptr=ptr-
        >next; }

    preptr->next=NULL;
    printf("\nElement popped from the stack is: %d\n",ptr->data);
    free(ptr);
}
}
void display()
{

    struct node *ptr;
    ptr=start;
    while(ptr!=NULL)
    {
        printf("\t%d",ptr->data);
        ptr=ptr->next;
    }
    printf("\n");
}

```

Output:



```
"C:\Users\Admin\Desktop\181" X + v

Enter the number to perform following operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
1
Enter the data
18

Enter the number to perform following operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
1
Enter the data
78

Enter the number to perform following operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
3
      78      18

Enter the number to perform following operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
2

Element popped from the stack is: 18
```

Lab Program 7:

WAP to Implement doubly link list with primitive operations

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

//Doubly Linked List

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

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

```
struct node
```

```
{
```

```
int data;
```

```
struct node *next;
```

```
struct node *prev;
```

```
};
```

```
struct node *start=NULL;
```

```
void create();
```

```
void insert();
```

```
void delete();
```

```
void display();
```

```
void main()
```

```
{
```

```
int option;
```

```
do
```

```
{
```

```
printf("\n***MAIN MENU***\n\n1.Create a doubly linked list.\n2.Insert at left\n3.Delete (specific value)\n4.Display\n5.Exit\n\nEnter an option: ");
```

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

```
switch(option)
```

```
{
```

```
case 1: create();
```

```
    printf("\nDoubly linked list created\n");
```

```
    break;
```

```
case 2: insert();
```

```
    printf("\nNode inserted\n");
```

```
    break;
```

```
case 3: delete();
```

```
    printf("\nNode deleted\n");
```

```
    break;
```

```
case 4: printf("\nElements in the doubly linked list\n");
```

```
    display();
```

```
    break;
```

```
} }while(option!
```

```
=5); }
```

```
void create()
```

```
{
```

```
struct node *new_node, *ptr;
```

```
int num;
```

```
printf("\nEnter -1 to end");
```

```
printf("\nEnter the data : ");
```

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

```
while(num!=-1)
```

```
{
```

```

if(start==NULL)
{
new_node=(struct node*)malloc(sizeof(struct node));
new_node->prev = NULL;
new_node->data = num;
new_node->next = NULL;
start=new_node;
}
else
{
ptr=start;
new_node=(struct node*)malloc(sizeof(struct node));
new_node->data=num;
while(ptr->next!=NULL)
ptr=ptr->next; ptr-
>next=new_node;
new_node->prev=ptr;
new_node-
>next=NULL; }
printf("\nEnter the data : ");
scanf("%d", &num);
}
}

```

```

void insert()
{
    struct node *new_node, *ptr;
    int pos, val, count = 0;

```

```
printf("\nEnter the data : ");
scanf("%d", &val);
printf("\nEnter the position before which the data has to be inserted:");
scanf("%d", &pos);
```

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

```
new_node->data = val;
```

```
ptr = start;
```

```
while (count < pos - 1 && ptr != NULL)
```

```
{
    ptr = ptr->next;
    count++;
}
```

```
if (count == pos - 1 && ptr != NULL)
```

```
{
    new_node->next = ptr;
```

```
    if (ptr->prev != NULL)
```

```
    {
        new_node->prev = ptr->prev;
        ptr->prev->next = new_node;
    }
```

```
    else
```

```
    {
        start = new_node;
        new_node->prev = NULL;
    }
```

```
    ptr->prev = new_node;
```

```

    }
else
{
    printf("Invalid position. Insertion failed.\n");
    free(new_node);
}
}

```

```

void delete()
{
    struct node *ptr;
    int num;
    printf("Enter the data to be deleted\n");
    scanf("%d", &num);

    ptr = start;
    while (ptr != NULL && ptr->data == num)
    {
        start = ptr->next;
        if (start != NULL)
            start->prev = NULL;
        free(ptr);
        ptr =
        start; }
    while (ptr != NULL && ptr->data != num)
        ptr = ptr->next;
    if (ptr == NULL)
    {
        printf("Data not found. Deletion failed.
\n"); }
}

```

```
else
{
if (ptr->prev != NULL) ptr-
>prev->next = ptr->next; if
(ptr->next != NULL) ptr-
>next->prev = ptr->prev;
free(ptr);
}
}
```

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


Output:

```
***MAIN MENU***

1.Create a doubly linked list.
2.Insert at left
3.Delete (specific value)
4.Display
5.Exit

Enter an option: 1

Enter -1 to end
Enter the data : 2

Enter the data : 4

Enter the data : 6

Enter the data : -1

Doubly linked list created

***MAIN MENU***

1.Create a doubly linked list.
2.Insert at left
3.Delete (specific value)
4.Display
5.Exit

Enter an option: 2

Enter the data : 8

Enter the position before which the data has to be inserted:2

Node inserted
```

```
***MAIN MENU***

1.Create a doubly linked list.
2.Insert at left
3.Delete (specific value)
4.Display
5.Exit

Enter an option: 4

Elements in the doubly linked list
2      8      4      6

***MAIN MENU***

1.Create a doubly linked list.
2.Insert at left
3.Delete (specific value)
4.Display
5.Exit

Enter an option: 3
Enter the data to be deleted
4

Node deleted

***MAIN MENU***

1.Create a doubly linked list.
2.Insert at left
3.Delete (specific value)
4.Display
5.Exit

Enter an option: 4

Elements in the doubly linked list
2      8      6

***MAIN MENU***

1.Create a doubly linked list.
2.Insert at left
3.Delete (specific value)
4.Display
5.Exit
```

Leetcode Program: Split Linked

List /**

* Definition for singly-linked list.

* struct ListNode {

* int val;

* struct ListNode *next;

* };

*/

/**

* Note: The returned array must be malloced, assume caller calls free().

*/

struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {

 struct ListNode* ptr=head;

 *returnSize=k;

 int count=0;

 while(ptr!=NULL)

 { count++;

 ptr=ptr->next;

 }

 int nums=count/k,a=count%k;

 struct ListNode **L=(struct ListNode**)calloc(k,sizeof(struct ListNode*));

 ptr=head;

 for(int i=0;i<k;i++){

 L[i] = ptr;

```

int segmentSize = nums + (a-- > 0 ? 1 : 0);

for (int j = 1; j < segmentSize; j++) {

    ptr = ptr->next;

}

if (ptr != NULL) {

    struct ListNode* next = ptr->next;

    ptr->next = NULL;

    ptr = next;

}

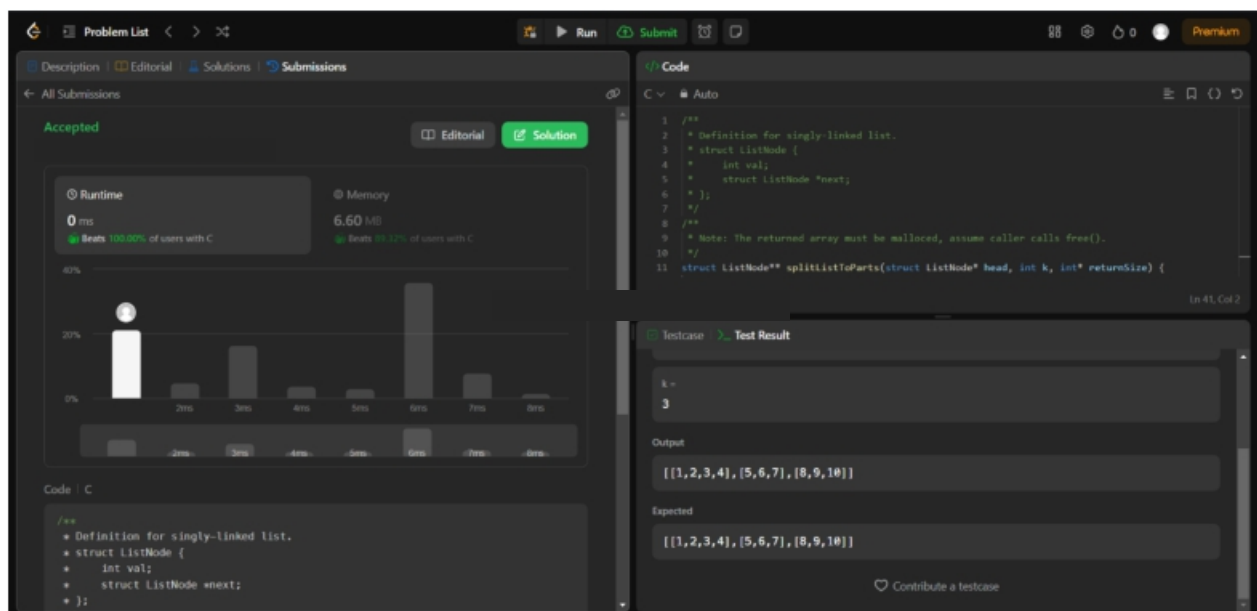
}

return L;

}

```

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.

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

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

```
struct node
```

```
{
```

```
    struct node *left;
```

```
    struct node *right;
```

```
    int data;
```

```
};
```

```
struct node *tree=NULL;
```

```
void create();
```

```
void pre(struct node *);
```

```
void post(struct node *);
```

```
void in(struct node *);
```

```
void main()
```

```
{
```

```
    int option;
```

```
    do
```

```
    {
```

```
        printf("\n\n***MAIN MENU***\n\n1.Create a binary search tree\n2.Preorder traversal\n3.Postorder traversal\n4.Inorder traversal\n5.Exit\n\nEnter an option: ");
```

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

```

switch(option)
{
case 1: create();
        printf("Binary search tree created\n\n");
        break;

case 2: printf("\nThe elements in the tree are\n");
        pre(tree);
        break;

case 3: printf("\nThe elements in the tree are\n");
        post(tree);
        break;

case 4: printf("\nThe elements in the tree are\n");
        in(tree);
        break;

}
}while(option!=5);
}

```

```

void create()
{
int val;
printf("\nEnter -1 to end");
printf("\nEnter the element : ");
scanf("%d",&val); while(val!
=-1)
{
struct node *ptr, *nodeptr, *parentptr;
ptr = (struct node*)malloc(sizeof(struct node));
ptr->data = val;

```

```

ptr->left = NULL;
ptr->right = NULL;
if(tree==NULL)
{
tree=ptr; tree-
>left=NULL; tree-
>right=NULL; }
else
{ parentptr=NULL
; nodeptr=tree;
while(nodeptr!=NULL)
{
parentptr=nodeptr;
if(val<nodeptr->data)
nodeptr=nodeptr->left;
else
nodeptr = nodeptr->right;
} if(val<parentptr-
>data) parentptr->left =
ptr; else
parentptr->right =
ptr; }
printf("\nEnter the element : ");
scanf("%d",&val);
}
}

```

```

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

```

```

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

```

```

void post(struct node *tree)
{
if(tree != NULL)
{
post(tree->left); post(tree-
>right); printf("%d\t",
tree->data);
}}

```

Output:

```
***MAIN MENU***
1.Create a binary search tree
2.Preorder traversal
3.Postorder traversal
4.Inorder traversal
5.Exit

Enter an option: 1

Enter -1 to end
Enter the element : 8

Enter the element : 1

Enter the element : 5

Enter the element : 3

Enter the element : 9

Enter the element : 4

Enter the element : 6

Enter the element : 7

Enter the element : -1
Binary search tree created
```

```
***MAIN MENU***

1.Create a binary search tree
2.Preorder traversal
3.Postorder traversal
4.Inorder traversal
5.Exit

Enter an option: 2

The elements in the tree are
8      1      5      3      4      6      7      9

***MAIN MENU***

1.Create a binary search tree
2.Preorder traversal
3.Postorder traversal
4.Inorder traversal
5.Exit
```



```
C:\Users\Admin\Desktop\1BNV X + v

Enter an option: 3

The elements in the tree are
4      3      7      6      5      1      9      8

***MAIN MENU***
1.Create a binary search tree
2.Preorder traversal
3.Postorder traversal
4.Inorder traversal
5.Exit

Enter an option: 4

The elements in the tree are
1      3      4      5      6      7      8      9

***MAIN MENU***
1.Create a binary search tree
2.Preorder traversal
3.Postorder traversal
4.Inorder traversal
5.Exit

Enter an option: 5

Process returned 5 (0x5)   execution time : 60.335 s
Press any key to continue.
|
```

Leetcode Program: Rotate

List /**

* Definition for singly-linked list.

* struct ListNode {

* int val;

* struct ListNode *next;

* };

*/

struct ListNode* rotateRight(struct ListNode* head, int k) {

struct ListNode *ptr,*ptr1;

int count=0,num;

if(head==NULL || head->next==NULL){

return head;

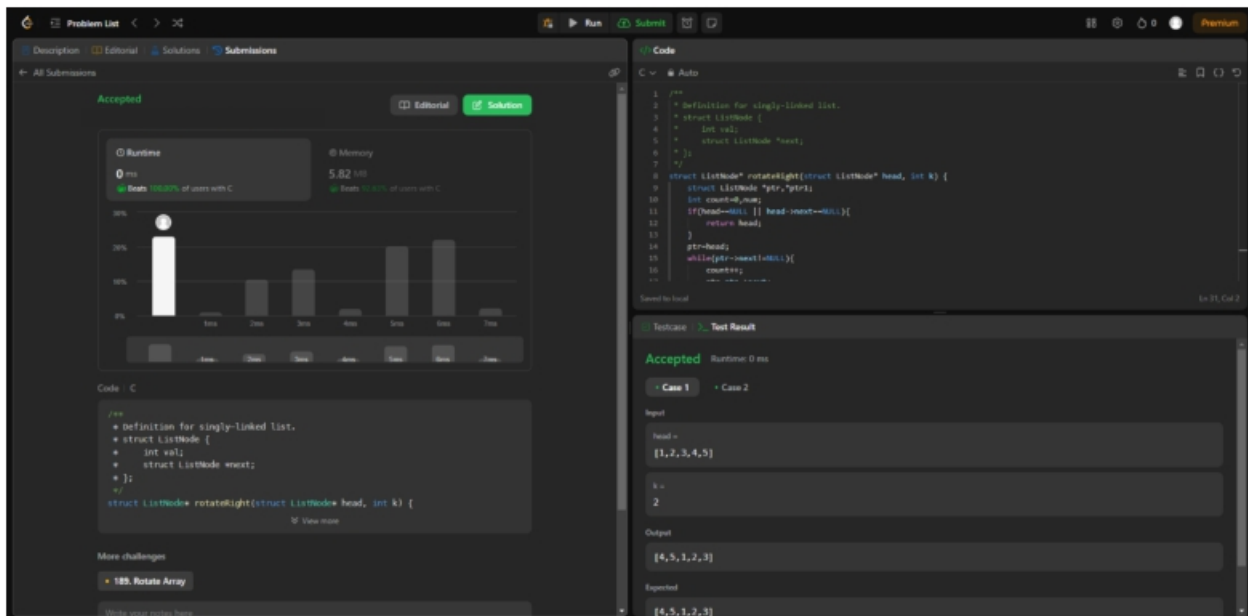
}

```

ptr=head;
while(ptr->next!=NULL){
    count++;
    ptr=ptr->next;
} num=k%
(count+1);
while(num--){
    ptr=head; while(ptr->next!
=NULL){ ptr1=ptr;
    ptr=ptr->next;
    }
    ptr->next=head;
    ptr1->next=NULL;
    head=ptr;
}
return head;
}

```

Output:



Lab Program 9:

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

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

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

```
#define MAX_VERTICES 50
```

```
typedef struct Graph_t {
```

```
    int V;
```

```
    int adj[MAX_VERTICES]
```

```
[MAX_VERTICES]; } Graph;
```

```
int DFS_V[50];
```

```
Graph* Graph_create(int V)
```

```
{
```

```
    Graph* g = malloc(sizeof(Graph));
```

```
    g->V = V;
```

```

    for (int i = 0; i <=V; i++) {
        for (int j = 0; j <=V; j++) {
            g->adj[i][j] = 0;
        }
    }

    return g;
}

void Graph_addEdge(Graph* g, int v, int w)
{
    g->adj[v][w] = 1;
    g->adj[w][v] = 1;
}

void BFS(Graph* g, int root){
    int visited[g->V+1];
    for(int i=0;i<=g->V;i++)
        visited[i]=0;

    int queue[g->V+1];
    int front=0,rear=0;
    visited[root]=1;

    queue[rear++]=root;
    while(front!=rear){
        root=queue[front++];
        printf("%d ",root);
        for(int i=0;i<=g->V;i++){
            if(g->adj[root][i]==1 && visited[i]!=1){
                visited[i]=1;

```

```

        queue[rear++]=i;
    }
}
}

}

int DFS(Graph *g,int root){
    for(int i=0;i<=g->V;i++){
        if(g->adj[root][i]==1 && DFS_V[i]!=1){
            DFS_V[i]=1;
            DFS(g,i);
        }
    }
    int count=0;
    for(int i=0;i<=g->V;i++)
        { if(DFS_V[i]==1)
            { count++;
            }
        }
    return count;
}

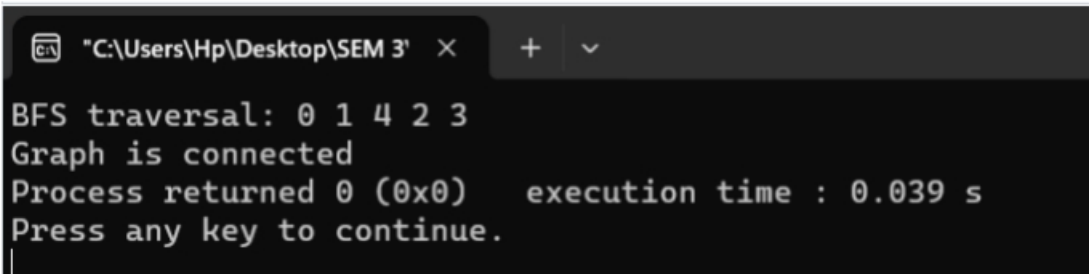
int main()
{

    Graph* g = Graph_create(4);
    Graph_addEdge(g, 0, 1);
    Graph_addEdge(g, 0, 4);
    Graph_addEdge(g, 1, 3);
    Graph_addEdge(g, 1, 2);

```

```
Graph_addEdge(g, 2, 3);
Graph_addEdge(g, 4, 3);
printf("BFS traversal: ");
BFS(g,0);
int count=DFS(g,0);
if(count==g->V+1){
    printf("\nGraph is connected");
}
else{
    printf("\nGraph is disconnected");
}
}
```

Output:



```
"C:\Users\Hp\Desktop\SEM 3" × + v
BFS traversal: 0 1 4 2 3
Graph is connected
Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

Lab Program 10:

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.

Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.

Let the keys in K and addresses in L are integers.

Design and develop a Program in C that uses Hash function $H: K \rightarrow L$ as $H(K)=K \bmod m$ (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 TABLE_SIZE 10

struct EmployeeRecord {
    int key;
    // Other fields of the employee record can be added here
};

struct EmployeeRecord *hashTable[TABLE_SIZE];

int hashFunction(int key) {
    return key % TABLE_SIZE;
}

void insert(struct EmployeeRecord *record) {
    int key = record->key;
    int index = hashFunction(key);
    int i = 0;

    while (i < TABLE_SIZE) {
        if (hashTable[index] == NULL) {
```

```

        hashTable[index] = record;

        printf("Inserted record with key %d at index %d\n", key, index);

        return;
    }

    i++;

    index = (hashFunction(key) + i) % TABLE_SIZE;
}

printf("HashTable is full. Unable to insert record with key %d\n", key);
}

```

```

struct EmployeeRecord* search(int key) {
    int index = hashFunction(key);
    int i = 0;

    while (i < TABLE_SIZE) {
        if (hashTable[index] != NULL && hashTable[index]->key == key)
            { printf("Record with key %d found at index %d\n", key, index);
              return hashTable[index];
            }

        i++;

        index = (hashFunction(key) + i) % TABLE_SIZE;
    }

    printf("Record with key %d not found in the HashTable\n", key);

    return NULL;
}

```

```

int main() {
    // Initialize hashTable with NULL pointers

    for (int i = 0; i < TABLE_SIZE; i++) {

```



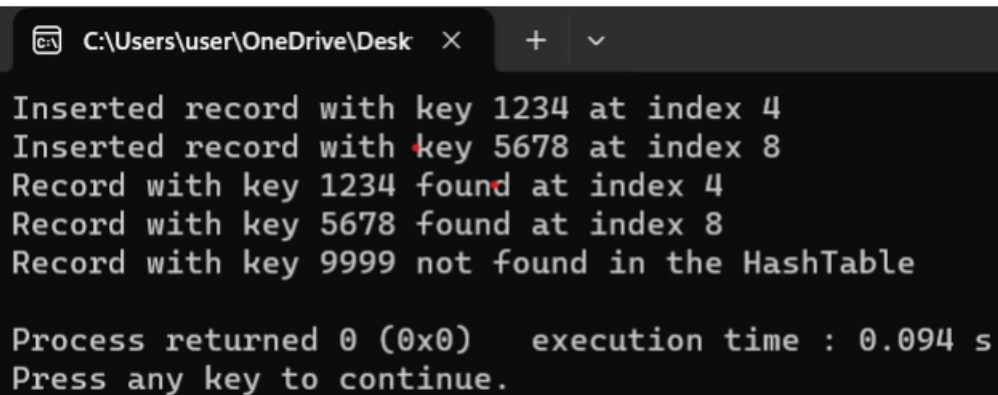
```
    hashTable[i] = NULL;
}
struct EmployeeRecord record1 = {1234}; // Example record with key 1234
struct EmployeeRecord record2 = {5678}; // Example record with key 5678

insert(&record1);
insert(&record2);

search(1234);
search(5678);
search(9999);

return 0;
}
```

Output:



```
C:\Users\user\OneDrive\Desk
Inserted record with key 1234 at index 4
Inserted record with key 5678 at index 8
Record with key 1234 found at index 4
Record with key 5678 found at index 8
Record with key 9999 not found in the HashTable

Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```