VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ROHIT KUMAWAT (1BF24CS256)

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
August-December 2025

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 Rohit Kumawat **(1BF24CS256)**, who is 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 2025-2026. 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.

Anusha S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

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

Lab program 1:

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

```
#include <stdio.h>
#define Stack_len 5

int stack[Stack_len];
int top = -1;

void push()
{
   int data;
   if (top == Stack_len - 1)
   {
      printf("Stack Overflow\n");
      return;
   }
   else
   {
      printf("Enter value to insert\n");
      scanf("%d", &data);
      stack[++top] = data;
   }
}

void pop()
{
   if (top == -1)
   {
      printf("StacK underflow\n");
      return;
   }
   else
   {
      printf("%d Removed\n", stack[top]);
      top--;
      return;
   }
}

void display()
{
   if(top==-1){
      printf("Stack is empty\n");
      return;
   }printf("Displaying Stack\n");
```

```
    for(int i=top;i>-1;i--){

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

int main()
{

    int n;

    while (1)
    {
        printf("Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit\n");
        scanf("%d", &n);
        switch (n)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            return 0;
        default:
            printf("Enter a valid input");
            break;
        }
    }

    return 0;
}
```

Output:

```
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\Lab Programn 1> cd "c:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab progr
n1 }
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
1
Enter value to insert
2
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
1
Enter value to insert
3
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
1
Enter value to insert
4
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
1
Enter value to insert
5
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
1
Enter value to insert
6
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
1
Stack Overflow
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
2
6 Removed
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
3
Displaying Stack
5
4
3
2
Enter 1 for push, 2 for pop , 3 for Display Stack & 4 for exit
4
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\Lab Programn 1>
```

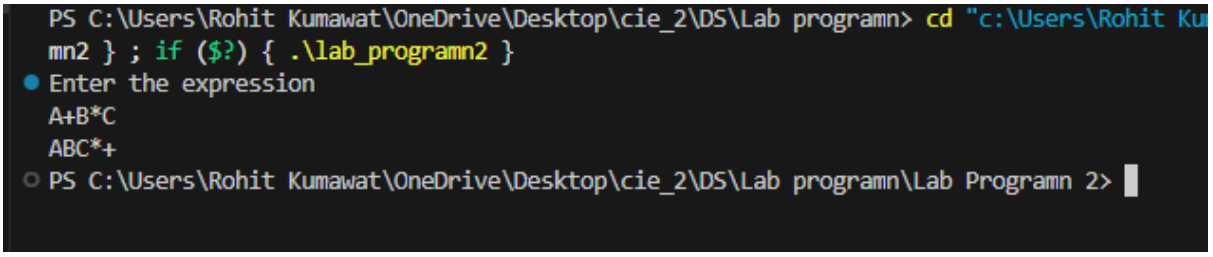
Lab program 2a:

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

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#define MAX 100

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

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

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

int presidence(char c){
   if(c=='^'){
      return 3;
   }
   if(c=='*' || c== '/'){
      return 2;
   }
   if(c=='+' || c=='-'){
      return 1;
   }
   return 0;
}
int match(char a,char b){
   if(a=='('  && b==')'){
      return 1;
   }
   if(a=='['  && b==']'){
      return 1;
   }
   if(a=='{'  && b=='}'){
      return 1;
   }
   return 0;
}
```

```
int main(){
   char infix[MAX];
   int check;
   int temp=0;
   printf("Enter the expression\n");
   gets(infix);
   for(int i =0;i<strlen(infix);i++){
      if(isalnum(infix[i])){
         printf("%c",infix[i]);
      }
      else if(infix[i]=='(' || infix[i]=='{' || infix[i] =='['){
         push(infix[i]);
      }
      else if(infix[i]==')' || infix[i]=='}' || infix[i] ==']'){
         while(!match(stack[top],infix[i])){
            printf("%c",pop());
         }
         pop();
      }
      else{
         while(top!=-1 && presidence(infix[i])<=presidence(stack[top])){
            printf("%c",pop());
         }
         push(infix[i]);
      }
   }
   while (top!=-1)
   {
      printf("%c",pop());
   }
   return 0;
}
```

Output:



Lab Programn 3a:

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

```
#include <stdio.h>
#define MAX 5
int queue[MAX];
int front = -1, end = -1;

void insert(int data)
{
   if (end == MAX - 1)
   {
      printf("Queue is FULL\n");
      return;
   }
   else
   {
      if (front == -1)
      {
         front = 0;
         end = 0;
         queue[end] = data;
         return;
      }
      queue[++end] = data;
   }
}

void delete()
{
   if (front == -1)
   {
      printf("Queue is Empty\n");
      return;
   }
   printf("%d Removed\n", queue[front]);
   if(front==end){
      front=end=-1;
   }
   else{
      front++;
   }
}

void display()
{
   if (front == -1)
   {
```

```
        printf("Queue is Empty");
        return;
    }
    printf("Printing Queue\n");
    for (int i = front; i <= end; i++)
    {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main()
{

    int value;
    int num;
    while(1){
    printf("Enter 1 for insert , 2 for delete , 3 for display,4 exit\n");
    scanf("%d", &value);
    switch (value)
    {
    case 1:
        printf("Enter value to insert\n");
        scanf("%d", &num);
        insert(num);
        break;
    case 2:
        delete();
        break;
    case 3:
        display();
        break;
    case 4:
        return 0;
    default:
        break;
    }}
    return 0;
}
```

Output:

```
n3 } ; if ($?) { .\lab_programn3 }
Enter 1 for insert , 2 for delete , 3 for display,4 exit
1
Enter value to insert
2
Enter 1 for insert , 2 for delete , 3 for display,4 exit
1
Enter value to insert
3
Enter 1 for insert , 2 for delete , 3 for display,4 exit

1
Enter value to insert
5
Enter 1 for insert , 2 for delete , 3 for display,4 exit
1
Enter value to insert
7
Enter 1 for insert , 2 for delete , 3 for display,4 exit
2
2 Removed
Enter 1 for insert , 2 for delete , 3 for display,4 exit
3
Printing Queue
3 5 7
Enter 1 for insert , 2 for delete , 3 for display,4 exit
4
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\lab programn3>
```

Lab Programn 3b:

b) 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 front = -1;
int rear = -1;
int queue[MAX];

void enque(int data)
{
   if (front == -1 && rear == -1)
   {
      front = rear = 0;
      queue[rear] = data;
      return;
   }
   if ((rear + 1) % MAX == front)
   {
      printf("Queue is FULL\n");
      return;
   }
   rear=(rear+1)%MAX;
```

```
    queue[rear]=data;
}

void dequeue()
{
    if (front == -1)
    {
        printf("Queue is empty\n");
        return;
    }
    if (front == rear)
    {
        front = rear = -1;
        return;
    }
    front = (front + 1) % MAX;
}

void display()
{
    if (front == -1)
    {
        printf("Queue is EMPTY\n");
        return;
    }

    printf("Printing QUEUE\n");

    int temp = front;
    while (1)
    {
        printf("%d ", queue[temp]);

        if (temp == rear)
            break;

        temp = (temp + 1) % MAX;
    }

    printf("\n");
}


int main()
{
    int value;
    int data;
    while (1)
    {
        printf("Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit\n");
```

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

        switch (value)
        {
        case 1:
            printf("Enter the data to insert\n");
            scanf("%d", &data);
            enque(data);
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            return 0;
        default:
            break;
        }
    }

    return 0;
}
```

Output:

```
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn> cd "c:\Users\Rohit Kumawat\One
mn3b }
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
1
Enter the data to insert
1
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
1
Enter the data to insert
2
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit

1
Enter the data to insert
3
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
2
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
3
Printing QUEUE
2 3
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
2
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
3
Printing QUEUE
3
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
3
Printing QUEUE
3
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
2
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
2
Queue is empty
Enter 1 for enque,2 for dequeue ,3 for display & 4 for exit
4
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\lab programn3>
```

Lab Programn 4a:

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 *head = NULL;

void createLinkedList(int n)
{
   struct Node *newNode, *temp;
   int data;
   if (head == NULL)
   {
      newNode = (struct Node *)malloc(sizeof(struct Node));
      printf("Enter the value \n");
      scanf("%d", &data);
      newNode->data = data;
      newNode->next = NULL;
      head = newNode;
   }
   temp = head;
   for (int i = 1; i < n; i++)
   {

      newNode = (struct Node *)malloc(sizeof(struct Node));
      printf("Enter the value \n");
      scanf("%d", &data);
      newNode->data = data;
      newNode->next = NULL;
      temp->next = newNode;
      temp=temp->next;
   }
}

void insertAtBeginning(int data)
{
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
```

```
}

void insertAtEnd(int data)
{
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node ));
   newNode->data = data;
   newNode->next = NULL;
   if (head == NULL)
   {
      head = newNode;
      return;
   }
   struct Node *temp = head;
   while (temp->next != NULL)
   {
      temp = temp->next;
   }
   temp->next = newNode;
}

void display()
{
   if (head == NULL)
   {
      printf("List is empty\n");
      return;
   }
   struct Node *temp = head;
   while (temp != NULL)
   {
      printf("%d ->", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}

void insertAtSpecific(int data , int pos){
   struct Node* newNode =  (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   if(pos<0){
      printf("Index out of Bound");
      return;
   }
   if(pos==1){
      insertAtBeginning(data);
      return;
   }
   else{
      struct Node * temp = head;
      for(int i=1;i<pos-1 && temp!=NULL;i++){
```

```
            temp=temp->next;
        }
        if(temp==NULL){
            free(newNode);
            return;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

int main(){
int choice, n, data, pos;

    while (1)
    {
        printf("\n---- LINKED LIST MENU ----\n");
        printf("1. Create Linked List\n");
        printf("2. Insert at Beginning\n");
        printf("3. Insert at End\n");
        printf("4. Insert at Specific Position\n");
        printf("5. Display Linked List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter number of nodes: ");
            scanf("%d", &n);
            createLinkedList(n);
            break;

        case 2:
            printf("Enter value to insert at beginning: ");
            scanf("%d", &data);
            insertAtBeginning(data);
            break;

        case 3:
            printf("Enter value to insert at end: ");
            scanf("%d", &data);
            insertAtEnd(data);
            break;

        case 4:
            printf("Enter value: ");
            scanf("%d", &data);
            printf("Enter position: ");
```

```
            scanf("%d", &pos);
            insertAtSpecific(data, pos);
            break;

        case 5:
            display();
            break;

        case 6:
            printf("Exiting...\n");
            exit(0);

        default:
            printf("Invalid choice! Try again.\n");
        }
    }

    return 0;
}
```

Output:

```
---- LINKED LIST MENU ----
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Specific Position
5. Display Linked List
6. Exit
Enter your choice: 1
Enter number of nodes: 4
Enter the value
1
Enter the value
2
Enter the value
3
Enter the value
4

---- LINKED LIST MENU ----
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Specific Position
5. Display Linked List
6. Exit
Enter your choice: 2
Enter value to insert at beginning: 2

---- LINKED LIST MENU ----
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Specific Position
5. Display Linked List
6. Exit
Enter your choice: 3
Enter value to insert at end: 9

---- LINKED LIST MENU ----
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Specific Position
5. Display Linked List
6. Exit
Enter your choice: 4
Enter value: 5
Enter position: 3

---- LINKED LIST MENU ----
1. Create Linked List
2. Insert at Beginning
```

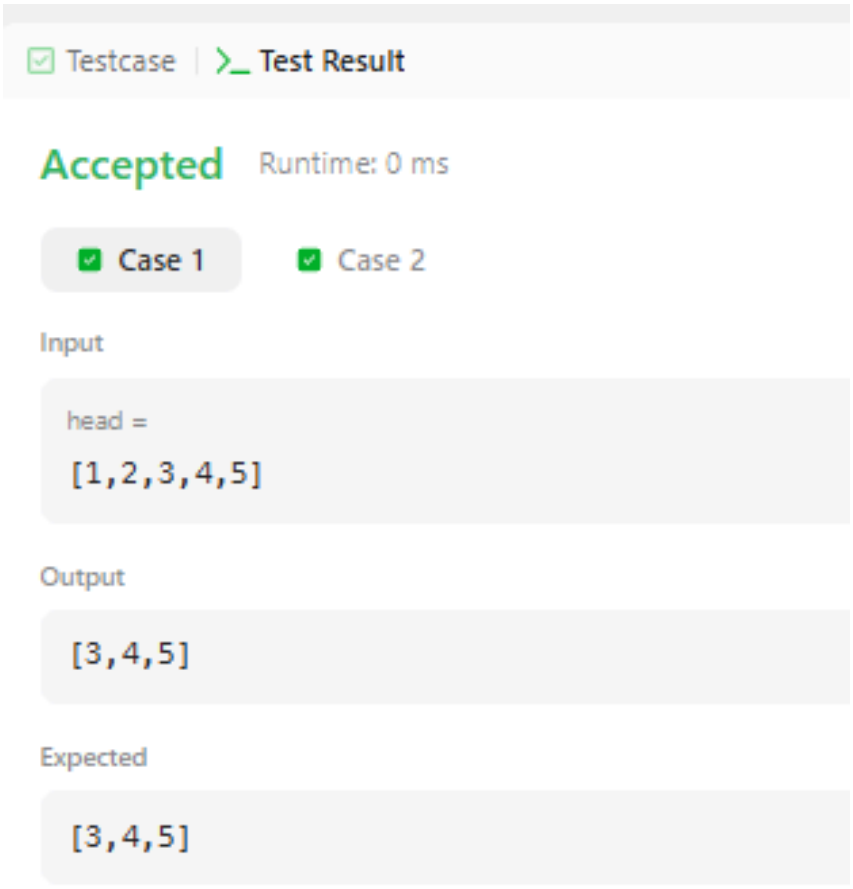
```
---- LINKED LIST MENU ----
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Specific Position
5. Display Linked List
6. Exit
Enter your choice: 4
Enter value: 5
Enter position: 3

---- LINKED LIST MENU ----
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Specific Position
5. Display Linked List
6. Exit
Enter your choice: 5
2 ->1 ->5 ->2 ->3 ->4 ->9 ->NULL

---- LINKED LIST MENU ----
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Specific Position
5. Display Linked List
6. Exit
Enter your choice: 6
Exiting...
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\lab programn 4>
```

LeetCode:
Program - Leetcode platform Link:
https://leetcode.com/problems/middle-of-the-linked-list/description/

```
struct ListNode* middleNode(struct ListNode* head) {
    struct ListNode *temp = head;
    int count = 0;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    int middleIndex = (count / 2);
    temp = head;
    while (middleIndex--) {
        temp = temp->next;
    }
```

```
    return temp;
}
```

Output:



Lab Programn 5(a):

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 *head = NULL;

void createLinkedList(int n)
{
   struct Node *newNode, *temp;
   int data;

   if (n <= 0)
      return;

   printf("Enter the value\n");
   scanf("%d", &data);

   newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   head = newNode;

   temp = head;

   for (int i = 1; i < n; i++)
   {
      newNode = (struct Node *)malloc(sizeof(struct Node));
      printf("Enter the value\n");
      scanf("%d", &data);
      newNode->data = data;
      newNode->next = NULL;
      temp->next = newNode;
      temp = newNode;
   }
}

void deleteFirst()
{
```

```
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }

    struct Node *temp = head;
    head = head->next;
    free(temp);
}

void deleteLast()
{
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }

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

    struct Node *temp = head;
    while (temp->next->next != NULL)
    {
        temp = temp->next;
    }

    free(temp->next);
    temp->next = NULL;
}

void deleteSpecific(int key)
{
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }

    if (head->data == key)
    {
        deleteFirst();
        return;
    }
```

```
    struct Node *temp = head;
    struct Node *prev = NULL;

    while (temp != NULL && temp->data != key)
    {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL)
    {
        printf("Element not found\n");
        return;
    }

    prev->next = temp->next;
    free(temp);
}

void display()
{
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }

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

int main()
{
int choice, n, key;

    while (1)
    {
        printf("\n---- SINGLY LINKED LIST MENU ----\n");
        printf("1. Create Linked List\n");
        printf("2. Delete First Element\n");
        printf("3. Delete Last Element\n");
        printf("4. Delete Specific Element\n");
        printf("5. Display Linked List\n");
        printf("6. Exit\n");
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter number of nodes: ");
            scanf("%d", &n);
            createLinkedList(n);
            break;

        case 2:
            deleteFirst();
            break;

        case 3:
            deleteLast();
            break;

        case 4:
            printf("Enter element to delete: ");
            scanf("%d", &key);
            deleteSpecific(key);
            break;

        case 5:
            display();
            break;

        case 6:
            printf("Exiting...\n");
            exit(0);

        default:
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
---- SINGLY LINKED LIST MENU ----
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specific Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter number of nodes: 3
Enter the value
1
Enter the value
2
Enter the value
3

---- SINGLY LINKED LIST MENU ----
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specific Element
5. Display Linked List
6. Exit
Enter your choice: 5
1 ->2 ->3 ->NULL

---- SINGLY LINKED LIST MENU ----
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specific Element
5. Display Linked List
6. Exit
Enter your choice: 2

---- SINGLY LINKED LIST MENU ----
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specific Element
5. Display Linked List
6. Exit
Enter your choice: 3

---- SINGLY LINKED LIST MENU ----
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specific Element
5. Display Linked List
6. Exit
Enter your choice: 4
```

```
Enter element to delete: 2

---- SINGLY LINKED LIST MENU ----
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specific Element
5. Display Linked List
6. Exit
Enter your choice: 5
List is empty

---- SINGLY LINKED LIST MENU ----
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specific Element
5. Display Linked List
6. Exit
Enter your choice: 6
Exiting...
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie 2\DS\Lab programn\lab programn 5>
```

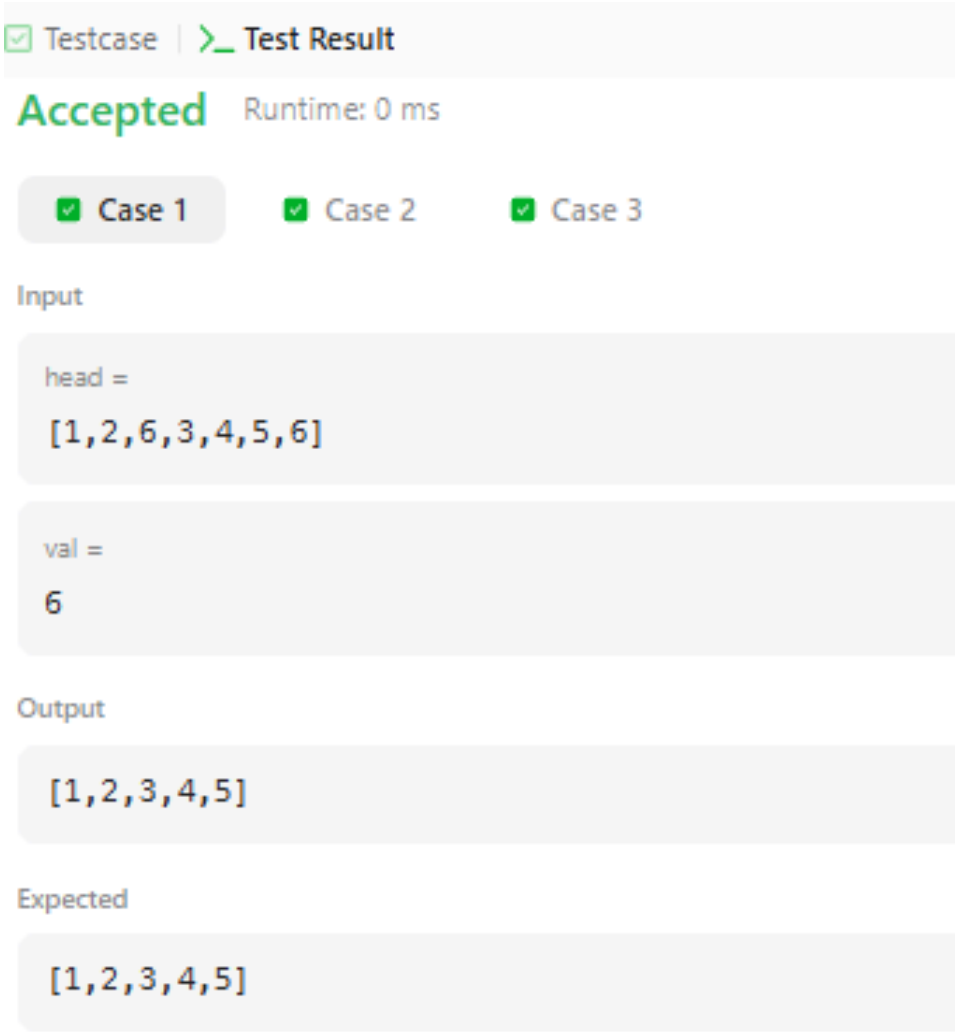
Lab Programn 5(b):

Program - Leetcode platform Link:
https://leetcode.com/problems/remove-linked-list-elements/description/

```
struct ListNode* removeElements(struct ListNode* head, int val) {

    struct ListNode *temp = head;
    struct ListNode *prev = NULL;

    while (temp != NULL) {
        if (temp->val == val) {
            if (prev == NULL) {
                head = temp->next;
                free(temp);
                temp = head;
            }
            else {
                prev->next = temp->next;
                free(temp);
                temp = prev->next;
            }
```

```
        }
        else {
            prev = temp;
            temp = temp->next;
        }
    }

    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.

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

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

struct Node* head1 = NULL;
struct Node* head2 = NULL;

struct Node* createLinked(int n){
   struct Node *newNode, *temp;
   int data;

   printf("Enter value of element 1:\n");
   scanf("%d",&data);

   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   struct Node *head = newNode;
   temp = head;

   for(int i = 2; i <= n; i++){
      printf("Enter value of element %d:\n", i);
      scanf("%d",&data);

      newNode = (struct Node*)malloc(sizeof(struct Node));
      newNode->data = data;
      newNode->next = NULL;

      temp->next = newNode;
      temp = newNode;
   }

   return head;
}

void display(struct Node* head){
   if(head == NULL){
      printf("List is empty\n");
      return;
   }
   struct Node* temp = head;
```

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

void sortList(struct Node* head){
    if(head == NULL){
        printf("List is empty\n");
        return;
    }

    int temp;
    for(struct Node* i = head; i != NULL; i = i->next){
        for(struct Node* j = i->next; j != NULL; j = j->next){
            if(i->data > j->data){
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }

    printf("List Sorted\n");
}

struct Node* reverse(struct Node* head){
    struct Node *curr = head, *prev = NULL, *next = NULL;

    while(curr != NULL){
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }

    printf("List Reversed\n");
    return prev;
}

struct Node* concatenate(struct Node* head1, struct Node* head2){
    if(head1 == NULL) return head2;
    if(head2 == NULL) return head1;

    struct Node* temp = head1;
    while(temp->next != NULL)
        temp = temp->next;
```

```
    temp->next = head2;

    printf("Lists Concatenated\n");
    return head1;
}

int main(){
    int choice, n1, n2;

    while(1){
        printf("\n1. Create List 1\n2. Create List 2\n3. Display List 1\n4. Display List 2\n5. Sort
List 1\n6. Sort List 2\n7. Reverse List 1\n8. Reverse List 2\n9. Concatenate\n10. Exit\n");
        printf("Enter choice: ");
        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter number of nodes: ");
                scanf("%d",&n1);
                head1 = createLinked(n1);
                break;

            case 2:
                printf("Enter number of nodes: ");
                scanf("%d",&n2);
                head2 = createLinked(n2);
                break;

            case 3:
                display(head1);
                break;

            case 4:
                display(head2);
                break;

            case 5:
                sortList(head1);
                break;

            case 6:
                sortList(head2);
                break;

            case 7:
                head1 = reverse(head1);
                break;

            case 8:
                head2 = reverse(head2);
```

```
            break;

        case 9:
            head1 = concatenate(head1, head2);
            printf("Final List: ");
            display(head1);
            break;

        case 10:
            return 0;

        default:
            printf("Invalid choice\n");
      }
   }
}
```

Output:

```
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn> cd "c:\Use
}
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate
10. Exit
Enter choice:  1
Enter number of nodes: 3
Enter value of element 1:
1
Enter value of element 2:
2
Enter value of element 3:
3

1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate
10. Exit
Enter choice: 2
Enter number of nodes: 4
Enter value of element 1:
5
Enter value of element 2:
6
Enter value of element 3:
7
Enter value of element 4:
8

1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate
10. Exit
Enter choice: 3
1 -> 2 -> 3 -> NULL

1. Create List 1
```

```
1 -> 2 -> 3 -> NULL

1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate
10. Exit
Enter choice: 4
5 -> 6 -> 7 -> 8 -> NULL

1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate
10. Exit
Enter choice: 9
Lists Concatenated
Final List: 1 -> 2 -> 3 -> 5 -> 6 -> 7 -> 8 -> NULL

1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate
10. Exit
Enter choice: 10
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\lab programn 6>
```

Lab program 6(b):
WAP to Implement Single Link List to simulate Stack & Queue Operations.

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

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

struct Node* top = NULL;
struct Node* front = NULL;
struct Node* rear = NULL;

void push(int x){
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = x;
   newNode->next = top;
   top = newNode;
}

void pop(){
   if(top == NULL){
      printf("Stack Underflow\n");
      return;
   }
   struct Node* temp = top;
   top = top->next;
   free(temp);
   printf("Popped\n");
}

void displayStack(){
   if(top == NULL){
      printf("Stack is Empty\n");
      return;
   }
   struct Node* temp = top;
   while(temp != NULL){
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}

void enqueue(int x){
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = x;
   newNode->next = NULL;
```

```
    if(front == NULL){
        front = rear = newNode;
    }
    else{
        rear->next = newNode;
        rear = newNode;
    }
}

void dequeue(){
    if(front == NULL){
        printf("Queue Underflow\n");
        return;
    }
    struct Node* temp = front;
    front = front->next;
    free(temp);
    if(front == NULL) rear = NULL;
    printf("Dequeued\n");
}

void displayQueue(){
    if(front == NULL){
        printf("Queue is Empty\n");
        return;
    }
    struct Node* temp = front;
    while(temp != NULL){
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main(){
    int choice, val;

    while(1){
        printf("\n1.Push\n2.Pop\n3.Display Stack\n4.Enqueue\n5.Dequeue\n6.Display
Queue\n7.Exit\n");
        printf("Enter choice: ");
        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter value: ");
                scanf("%d",&val);
                push(val);
                break;
```

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

            case 3:
                displayStack();
                break;

            case 4:
                printf("Enter value: ");
                scanf("%d",&val);
                enqueue(val);
                break;

            case 5:
                dequeue();
                break;

            case 6:
                displayQueue();
                break;

            case 7:
                return 0;

            default:
                printf("Invalid choice\n");
        }
    }
}
```

Output:

```
1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 1
Enter value: 1

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 1
Enter value: 2

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 2
Popped

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 3
1 -> NULL

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 4
Enter value: 4

1.Push
2.Pop
```

```
1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 4
Enter value: 4

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 4
Enter value: 6

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 5
Dequeued

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 6
6 -> NULL

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 7
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\lab programn 6>
```

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

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

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

struct Node *head = NULL, *tail = NULL;

void createlist(int n) {
   int i, data;
   struct Node *newNode;

   for (i = 1; i <= n; i++) {
      printf("Enter data for node %d: ", i);
      scanf("%d", &data);

      newNode = (struct Node*)malloc(sizeof(struct Node));
      newNode->data = data;
      newNode->prev = newNode->next = NULL;

      if (head == NULL) {
         head = tail = newNode;
      } else {
         tail->next = newNode;
         newNode->prev = tail;
         tail = newNode;
      }
   }
}

void insertAtFront(int data) {
   struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = head;

   if (head == NULL)
      head = tail = newNode;
   else {
      head->prev = newNode;
      head = newNode;
```

```
    }
}

void insertAtEnd(int data) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = tail;

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

void insertAtPosition(int data, int pos) {
    int i;
    struct Node *temp = head;

    if (pos == 1) {
        insertAtFront(data);
        return;
    }

    for (i = 1; i < pos - 1 && temp != NULL; i++)
        temp = temp->next;

    if (temp == NULL || temp->next == NULL) {
        insertAtEnd(data);
        return;
    }

    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;

    newNode->next = temp->next;
    newNode->prev = temp;
    temp->next->prev = newNode;
    temp->next = newNode;
}

void deleteAtFront() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }

    struct Node *temp = head;
```

```
    head = head->next;

    if (head != NULL)
        head->prev = NULL;
    else
        tail = NULL;

    free(temp);
}

void deleteAtEnd() {
    if (tail == NULL) {
        printf("List is empty!\n");
        return;
    }

    struct Node *temp = tail;
    tail = tail->prev;

    if (tail != NULL)
        tail->next = NULL;
    else
        head = NULL;

    free(temp);
}

void deleteByValue(int value) {
    struct Node *temp = head;

    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }

    while (temp != NULL && temp->data != value)
        temp = temp->next;

    if (temp == NULL) {
        printf("Value not found!\n");
        return;
    }

    if (temp == head)
        deleteAtFront();
    else if (temp == tail)
        deleteAtEnd();
    else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
```

```
        free(temp);
    }
}

void display() {
    struct Node *temp = head;

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

int main() {
    int choice, data, n, pos, value;

    while (1) {
        printf("\n---DOUBLY LINKED LIST MENU---\n");
        printf("1. Create List\n");
        printf("2. Insert at Front\n");
        printf("3. Insert at End\n");
        printf("4. Insert at Position\n");
        printf("5. Delete at Front\n");
        printf("6. Delete at End\n");
        printf("7. Delete by Value\n");
        printf("8. Display List\n");
        printf("9. Exit\n");
        printf("Choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter number of nodes: ");
            scanf("%d", &n);
            createlist(n);
            break;

        case 2:
            printf("Enter data: ");
            scanf("%d", &data);
            insertAtFront(data);
            break;

        case 3:
            printf("Enter data: ");
            scanf("%d", &data);
            insertAtEnd(data);
            break;
```

```
        case 4:
            printf("Enter data: ");
            scanf("%d", &data);
            printf("Enter position: ");
            scanf("%d", &pos);
            insertAtPosition(data, pos);
            break;

        case 5:
            deleteAtFront();
            break;

        case 6:
            deleteAtEnd();
            break;

        case 7:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            deleteByValue(value);
            break;

        case 8:
            display();
            break;

        case 9:
            exit(0);

        default:
            printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

Output:

```
---DOUBLY LINKED LIST MENU---
1. Create List
2. Insert at Front
3. Insert at End
4. Insert at Position
5. Delete at Front
6. Delete at End
7. Delete by Value
8. Display List
9. Exit
Choice: 1
Enter number of nodes: 3
Enter data for node 1: 1
Enter data for node 2: 2
Enter data for node 3: 3

---DOUBLY LINKED LIST MENU---
1. Create List
2. Insert at Front
3. Insert at End
4. Insert at Position
5. Delete at Front
6. Delete at End
7. Delete by Value
8. Display List
9. Exit
Choice: 1
Enter number of nodes: 2
Enter data for node 1: 2
Enter data for node 2: 3

---DOUBLY LINKED LIST MENU---
1. Create List
2. Insert at Front
3. Insert at End
4. Insert at Position
5. Delete at Front
6. Delete at End
7. Delete by Value
8. Display List
9. Exit
Choice: 5

---DOUBLY LINKED LIST MENU---
1. Create List
2. Insert at Front
3. Insert at End
4. Insert at Position
5. Delete at Front
6. Delete at End
7. Delete by Value
8. Display List
```

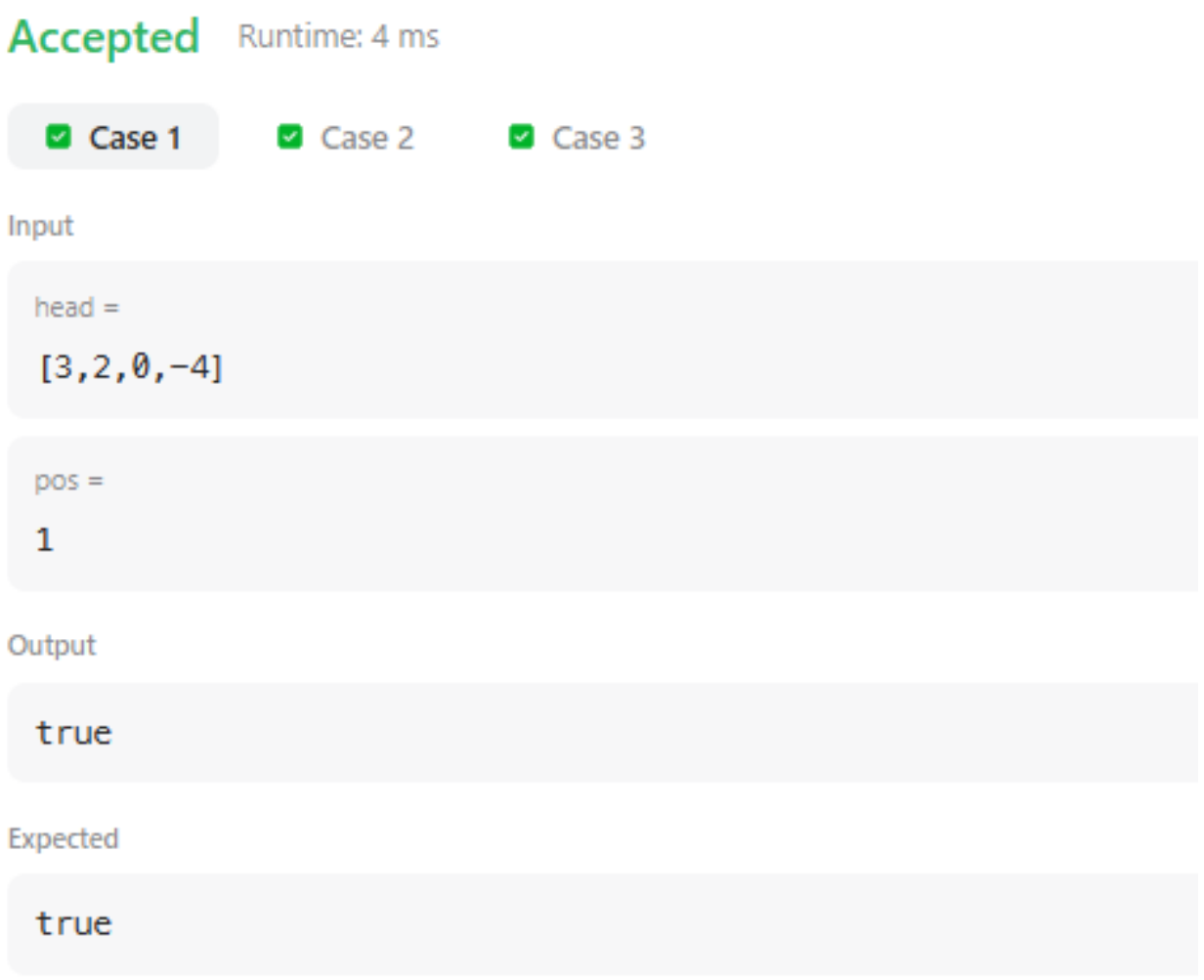
```
8. Display List
9. Exit
Choice: 7
Enter value to delete: 2

---DOUBLY LINKED LIST MENU---
1. Create List
2. Insert at Front
3. Insert at End
4. Insert at Position
5. Delete at Front
6. Delete at End
7. Delete by Value
8. Display List
9. Exit
Choice: 8
List : 3 <-> 2 <-> 3 <-> NULL

---DOUBLY LINKED LIST MENU---
1. Create List
2. Insert at Front
3. Insert at End
4. Insert at Position
5. Delete at Front
6. Delete at End
7. Delete by Value
8. Display List
9. Exit
Choice: 9
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\lab programn 7>
```

Lab program 7(b):

Program - Leetcode platform Link:
https://leetcode.com/problems/linked-list-cycle/description/

```
bool hasCycle(struct ListNode *head) {
    struct ListNode* visited[10001];
    int i = 0;
    while (head != NULL) {
        for (int j = 0; j < i; j++) {
            if (visited[j] == head)
                return true;
        }

        visited[i++] = head;
        head = head->next;
    }
    return false;
}
```

Output:


Accepted
Runtime: 4 ms
Case 1
Case 2
Case 3
Input
head =
[3,2,0,-4]
pos =
1
Output
true
Expected
true

Lab program 8(a):

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

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

struct node {
   int data;
   struct node *left, *right;
};

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

struct node* insert(struct node *root, int value) {
   if (root == NULL) {
      return createNode(value);
   }
   if (value < root->data) {
      root->left = insert(root->left, value);
   } else if (value > root->data) {
      root->right = insert(root->right, value);
   }
   return root;
}

void inorder(struct node *root) {
   if (root == NULL) return;
   inorder(root->left);
   printf("%d ", root->data);
   inorder(root->right);
}

void preorder(struct node *root) {
   if (root == NULL) return;
   printf("%d ", root->data);
   preorder(root->left);
   preorder(root->right);
}

void postorder(struct node *root) {
```

```
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

void display(struct node *root) {
    printf("BST Elements (Inorder): ");
    inorder(root);
    printf("\n");
}

int main() {
    struct node *root = NULL;
    int choice, value;

    while (1) {
        printf("\n--- BINARY SEARCH TREE MENU ---\n");
        printf("1. Insert into BST\n");
        printf("2. Inorder Traversal\n");
        printf("3. Preorder Traversal\n");
        printf("4. Postorder Traversal\n");
        printf("5. Display BST\n");
        printf("6. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("Inorder Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
                printf("Preorder Traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder Traversal: ");
                postorder(root);
                printf("\n");
                break;
            case 5:
                display(root);
```

```
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice! Try again.\n");
      }
   }
   return 0;
}
```

Output:

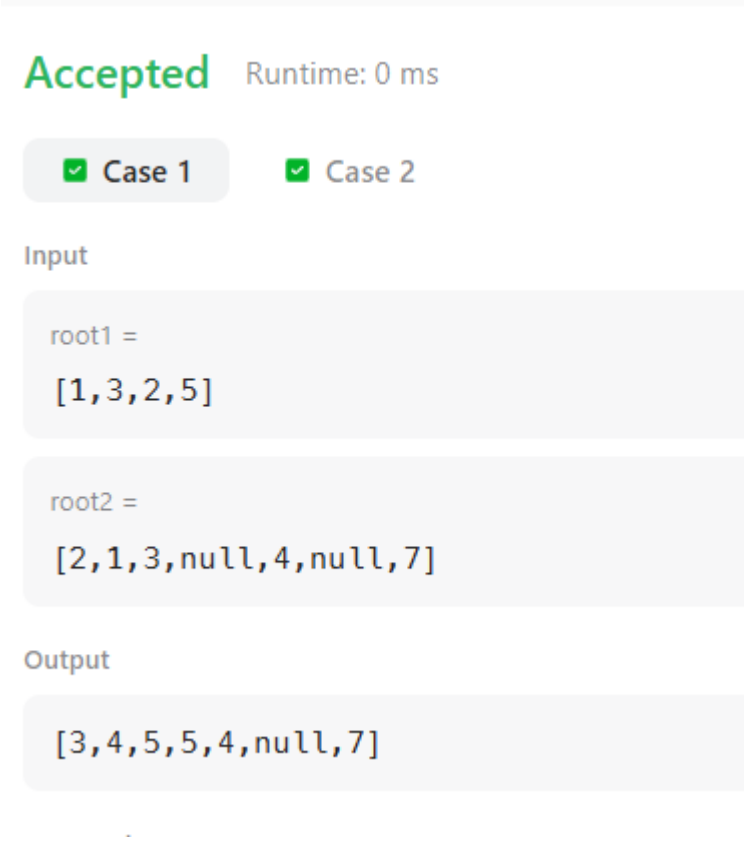
```
--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 1
Enter value to insert: 2

--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 1
Enter value to insert: 7

--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 1
Enter value to insert: 9

--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 1
Enter value to insert: 5

--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 2
Inorder Traversal: 2 5 7 9

--- BINARY SEARCH TREE MENU ---
1. Insert into BST
```

```
--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 3
Preorder Traversal: 2 7 5 9

--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 4
Postorder Traversal: 5 9 7 2

--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 5
BST Elements (Inorder): 2 5 7 9

--- BINARY SEARCH TREE MENU ---
1. Insert into BST
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter choice: 6
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn\lab programn 8>
```

Lab program 8(b):
Program - Leetcode platform Link:
https://leetcode.com/problems/merge-two-binary-trees/description/

```
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
    if (root1 == NULL) return root2;
    if (root2 == NULL) return root1;

    root1->val = root1->val + root2->val;
    root1->left = mergeTrees(root1->left, root2->left);
    root1->right = mergeTrees(root1->right, root2->right);
    return root1;
}
```

Output:

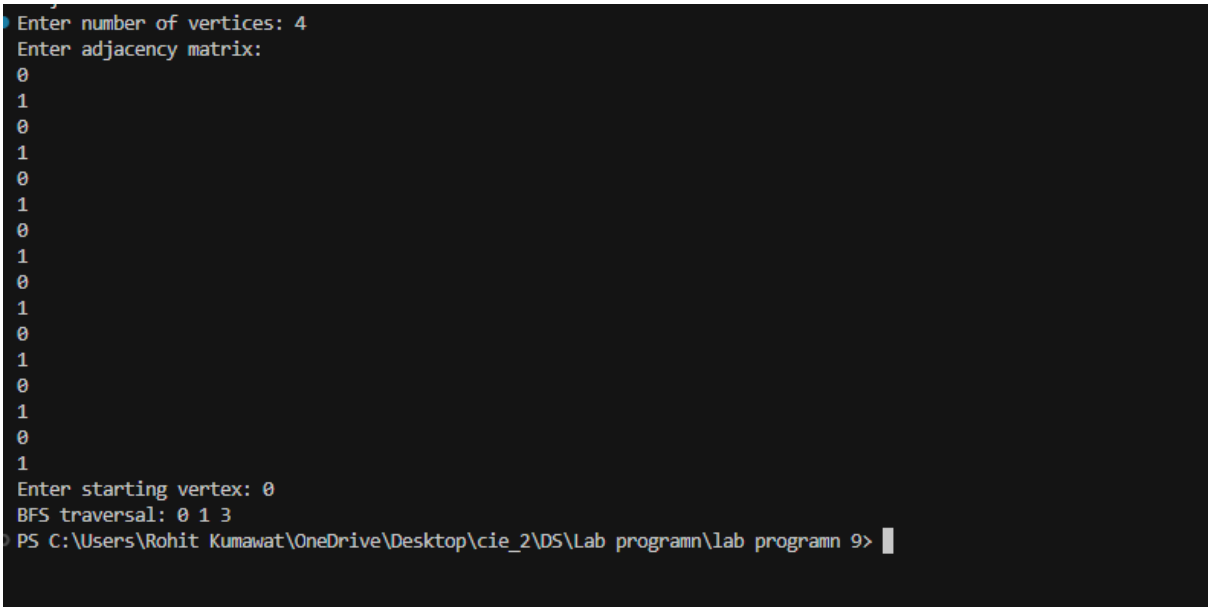

Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root1 =
[1,3,2,5]
root2 =
[2,1,3,null,4,null,7]
Output
[3,4,5,5,4,null,7]

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

```
#include <stdio.h>

int graph[20][20], visited[20], n;

void BFS(int start)
{
   int queue[20], front = 0, rear = 0;

   visited[start] = 1;
   queue[rear++] = start;

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

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

int main()
{
   int start;

   printf("Enter number of vertices: ");
   scanf("%d", &n);
```

```
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

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

    printf("Enter starting vertex: ");
    scanf("%d", &start);

    printf("BFS traversal: ");
    BFS(start);

    return 0;
}
```

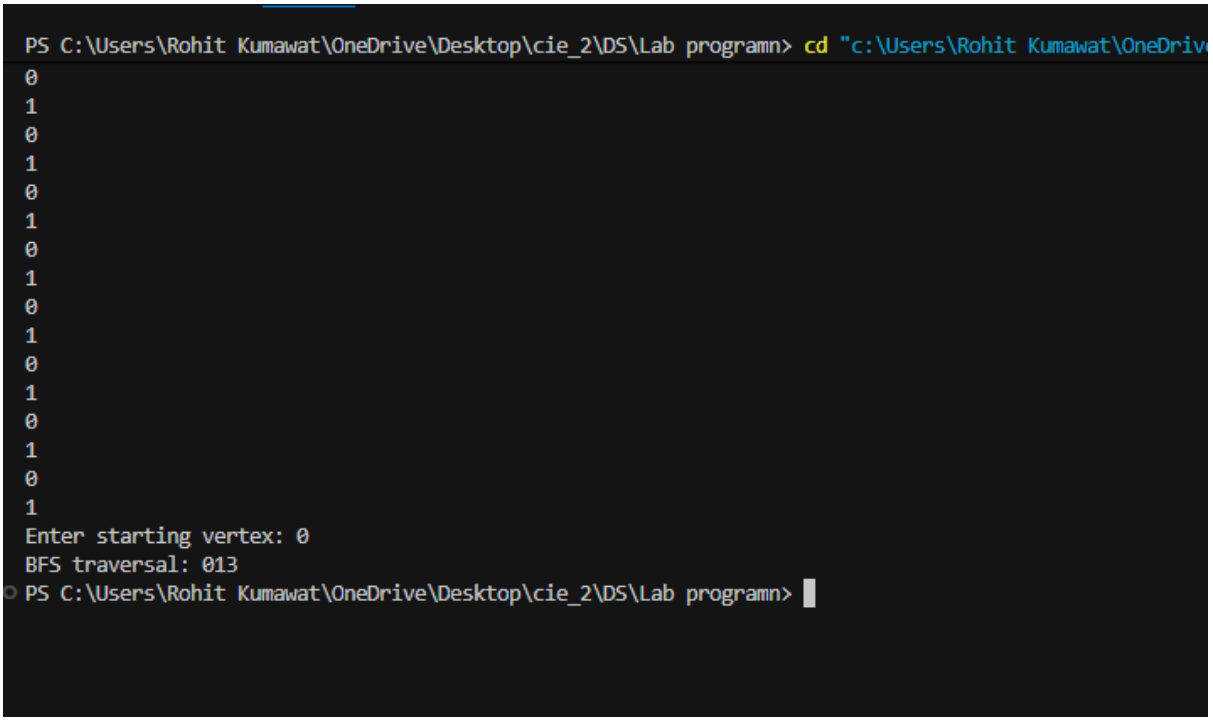
Output:



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

```
#include<stdio.h>
# define  max 10

int visited[max];
int adj[max][max];
int n;

void DFS(int v)
{
   visited[v]=1;
   printf("%d",v);

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

   }
}

int main()
{
   int start;

   printf("Enter number of vertices: ");
   scanf("%d", &n);

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

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

   printf("Enter starting vertex: ");
   scanf("%d", &start);

   printf("BFS traversal: ");
   DFS(start);

   return 0;
}
```

Output:


PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn> cd "c:\Users\Rohit Kumawat\OneDriv
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
Enter starting vertex: 0
BFS traversal: 013
PS C:\Users\Rohit Kumawat\OneDrive\Desktop\cie_2\DS\Lab programn>

Lab program 10:

Given a File of N employee records with a set K of Keys(4 digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include <stdio.h>

#define MAX 20

int hashTable[MAX];
int m;

void insert(int key)
{
   int index = key % m;
   if (hashTable[index] == -1)
   {
      hashTable[index] = key;
   }
   else
   {
      int i = 1;
      while (hashTable[(index + i) % m] != -1)
      {
         i++;
      }
      hashTable[(index + i) % m] = key;
   }
}

void display()
{
   printf("\nHash Table:\n");
   for (int i = 0; i < m; i++)
   {
      if (hashTable[i] != -1)
      {
         printf("Address %d: %d\n", i, hashTable[i]);
      }
      else
      {
         printf("Address %d: Empty\n", i);
      }
   }
}
```

```
int main()
{
   int n, key;

   printf("Enter size of hash table (m): ");
   scanf("%d", &m);

   printf("Enter number of employee records: ");
   scanf("%d", &n);

   for (int i = 0; i < m; i++)
   {
      hashTable[i] = -1;
   }

   printf("Enter %d employee keys (4-digit):\n", n);
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &key);
      insert(key);
   }

   display();
   return 0;
}
```

Output: