VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Govind Singh(1BM23CS102)

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
September 2024-January 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 **Govind Singh (1BM23CS102)**, 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 2024-25. 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.

Dr. Selva Kumar S	**Dr. Kavitha Sooda**
Associate Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course outcomes:

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

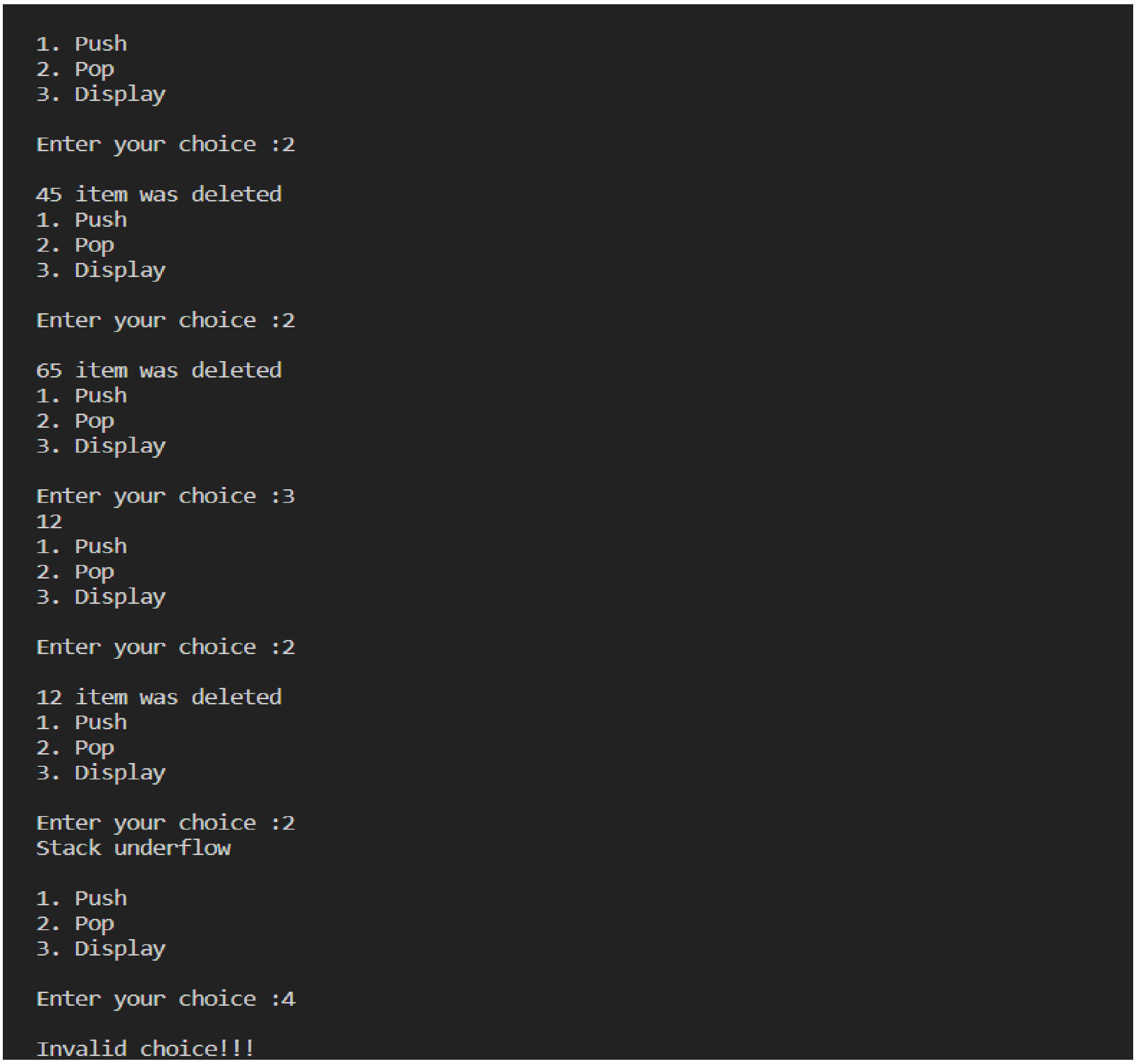
Lab program 1:

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

```
#include <stdio.h>
#include<stdlib.h>
#define STACK_SIZE 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:



```
1. Push
2. Pop
3. Display

Enter your choice :2

45 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2

65 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :3
12
1. Push
2. Pop
3. Display

Enter your choice :2

12 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2
Stack underflow

1. Push
2. Pop
3. Display

Enter your choice :4

Invalid choice!!!
```

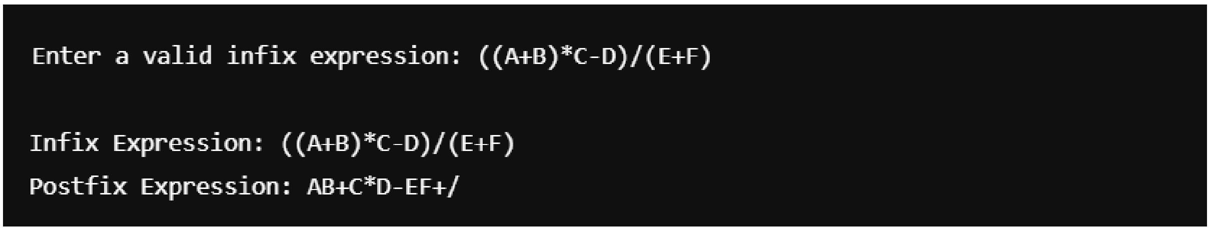
Lab program 2:

-

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

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

#define MAX 100

typedef struct {
    char data[MAX];
    int top;
} Stack;

void push(Stack *stack, char c) {
    stack->data[++stack->top] = c;
}

char pop(Stack *stack) {
    return stack->data[stack->top--];
}

char peek(Stack *stack) {
    return stack->data[stack->top];
}

int isEmpty(Stack *stack) {
    return stack->top == -1;
}

int precedence(char operator) {
    switch (operator) {
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 2;
        default: return -1;
    }
}

void infixToPostfix(char *infix, char *postfix) {
    Stack stack;
    stack.top = -1;
    int i, j = 0;
    char c;

    for (i = 0; infix[i] != '\0'; i++) {
        c = infix[i];
        if (isalnum(c)) {
            postfix[j++] = c;
        } else if (c == '(') {
            push(&stack, c);
        } else if (c == ')') {
            while (!isEmpty(&stack) && peek(&stack) != '(') {
                postfix[j++] = pop(&stack);
            }
            pop(&stack);
        } else {
            while (!isEmpty(&stack) && precedence(peek(&stack)) >=
precedence(c)) {
                postfix[j++] = pop(&stack);
            }
            push(&stack, c);
        }
    }

    while (!isEmpty(&stack)) {
        postfix[j++] = pop(&stack);
    }

    postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter a valid infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Postfix Expression: %s\n", postfix);
    return 0;
}
```

output



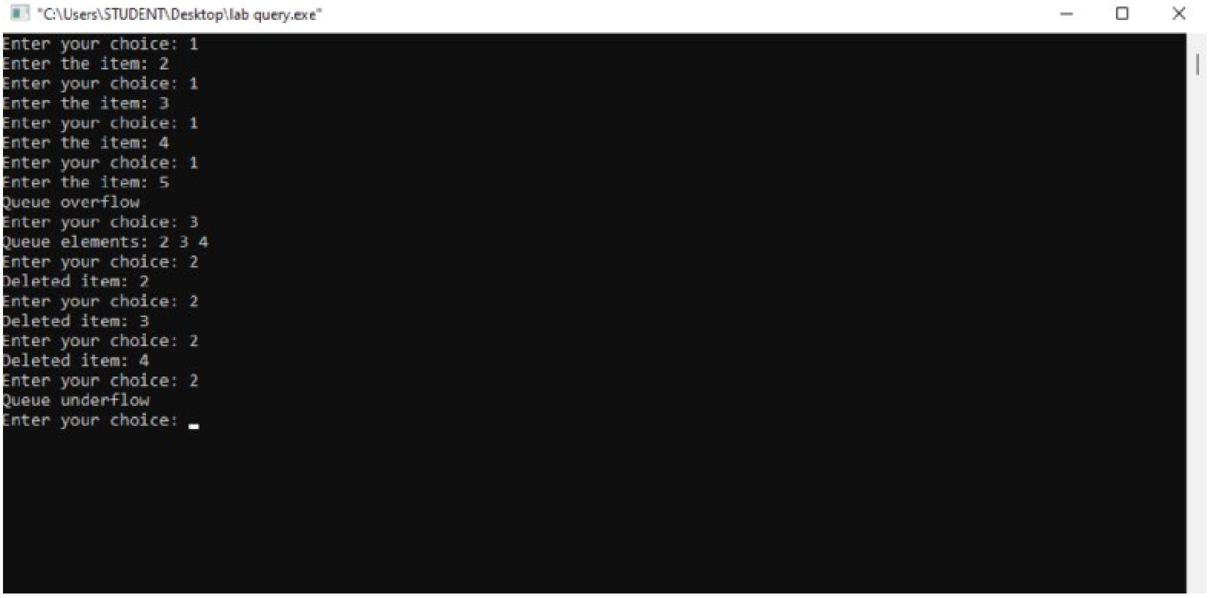
```
Enter a valid infix expression: ((A+B)*C-D)/(E+F)


Infix Expression: ((A+B)*C-D)/(E+F)
Postfix Expression: AB+C*D-EF+/
```

Lab program 3:

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

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

#define MAX 5

typedef struct {
    int data[MAX];
    int front;
    int rear;
} Queue;

void initializeQueue(Queue *q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(Queue *q) {
    return (q->rear + 1) % MAX == q->front;
}

int isEmpty(Queue *q) {
    return q->front == -1;
}

void insert(Queue *q, int value) {
    if (isFull(q)) {
        printf("\033[1;31mQueue Overflow! Cannot insert %d.\033[0m\n",
value);
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear = (q->rear + 1) % MAX;
    q->data[q->rear] = value;
    printf("\033[1;32mInserted %d into the queue.\033[0m\n", value);
}

void delete(Queue *q) {
    if (isEmpty(q)) {
        printf("\033[1;31mQueue Underflow! Queue is empty.\033[0m\n");
        return;
    }
    printf("\033[1;33mDeleted %d from the queue.\033[0m\n", q->data[q-
>front]);
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front = (q->front + 1) % MAX;
    }
}

void display(Queue *q) {
    if (isEmpty(q)) {
        printf("\033[1;31mQueue is empty.\033[0m\n");
        return;
    }
    printf("\033[1;34mQueue elements:\033[0m ");
    int i = q->front;
    while (1) {
        printf("%d ", q->data[i]);
        if (i == q->rear) break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);
    int choice, value;

    while (1) {
        printf("\n\033[1;36mQueue Operations:\033[0m\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(&q, value);
                break;
            case 2:
                delete(&q);
                break;
            case 3:
                display(&q);
                break;
            case 4:
                printf("\033[1;35mExiting program.\033[0m\n");
                exit(0);
            default:
                printf("\033[1;31mInvalid choice. Try again.\033[0m\n");
        }
    }
    return 0;
}
```

OUTPUT



```
Enter your choice: 1
Enter the item: 2
Enter your choice: 1
Enter the item: 3
Enter your choice: 1
Enter the item: 4
Enter your choice: 1
Enter the item: 5
Queue overflow
Enter your choice: 3
Queue elements: 2 3 4
Enter your choice: 2
Deleted item: 2
Enter your choice: 2
Deleted item: 3
Enter your choice: 2
Deleted item: 4
Enter your choice: 2
Queue underflow
Enter your choice: _
```

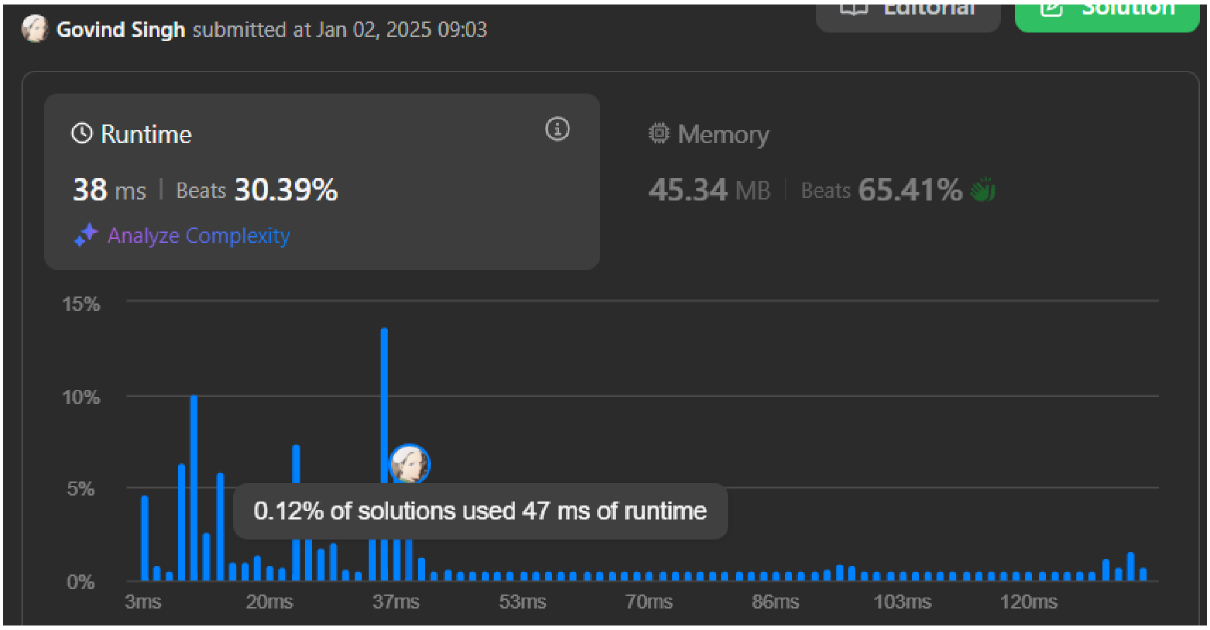
<u>LEETCODE PROBLEM:</u>

3b) Remove all adjacent duplicates in a string

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

```
#define MAX 1000

typedef struct {
    char data[MAX];
    int top;
} Stack;

void push(Stack *stack, char c) {
    stack->data[++stack->top] = c;
}

char pop(Stack *stack) {
    if (stack->top == -1) return '\0';
    return stack->data[stack->top--];
}

char peek(Stack *stack) {
    if (stack->top == -1) return '\0';
    return stack->data[stack->top];
}

int isEmpty(Stack *stack) {
    return stack->top == -1;
}

void removeAdjacentDuplicates(char *input, char *output) {
    Stack stack;
    stack.top = -1;
    int i;

    for (i = 0; input[i] != '\0'; i++) {
        if (!isEmpty(&stack) && peek(&stack) == input[i]) {
            pop(&stack);
        } else {
            push(&stack, input[i]);
        }
    }

    int j = 0;
    while (!isEmpty(&stack)) {
        output[j++] = pop(&stack);
    }
    output[j] = '\0';

    int len = strlen(output);
    for (i = 0; i < len / 2; i++) {
        char temp = output[i];
        output[i] = output[len - i - 1];
        output[len - i - 1] = temp;
    }
}

int main() {
    char input[MAX], output[MAX];
    printf("Enter a string: ");
    scanf("%s", input);

    removeAdjacentDuplicates(input, output);

    printf("Result after removing adjacent duplicates: %s\n", output);
    return 0;
}
```

OUTPUT



Govind Singh submitted at Jan 02, 2025 09:03

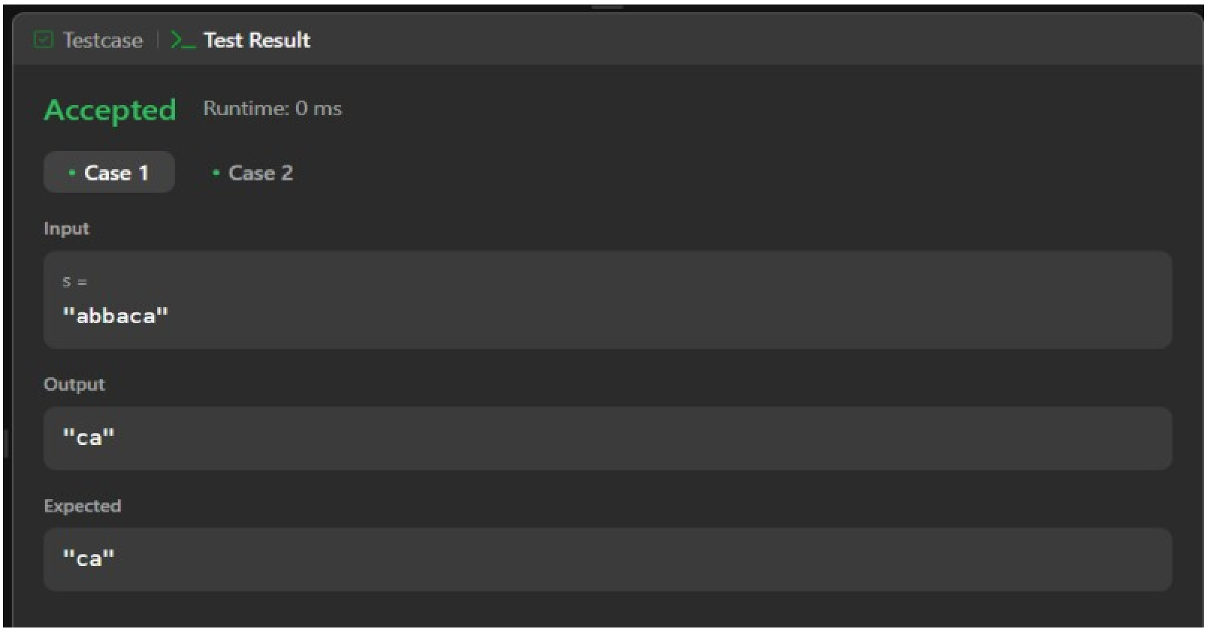
Runtime

38 ms | Beats 30.39%

Memory

45.34 MB | Beats 65.41%

0.12% of solutions used 47 ms of runtime



Accepted Runtime: 0 ms

Input

s =
"abbaca"

Output

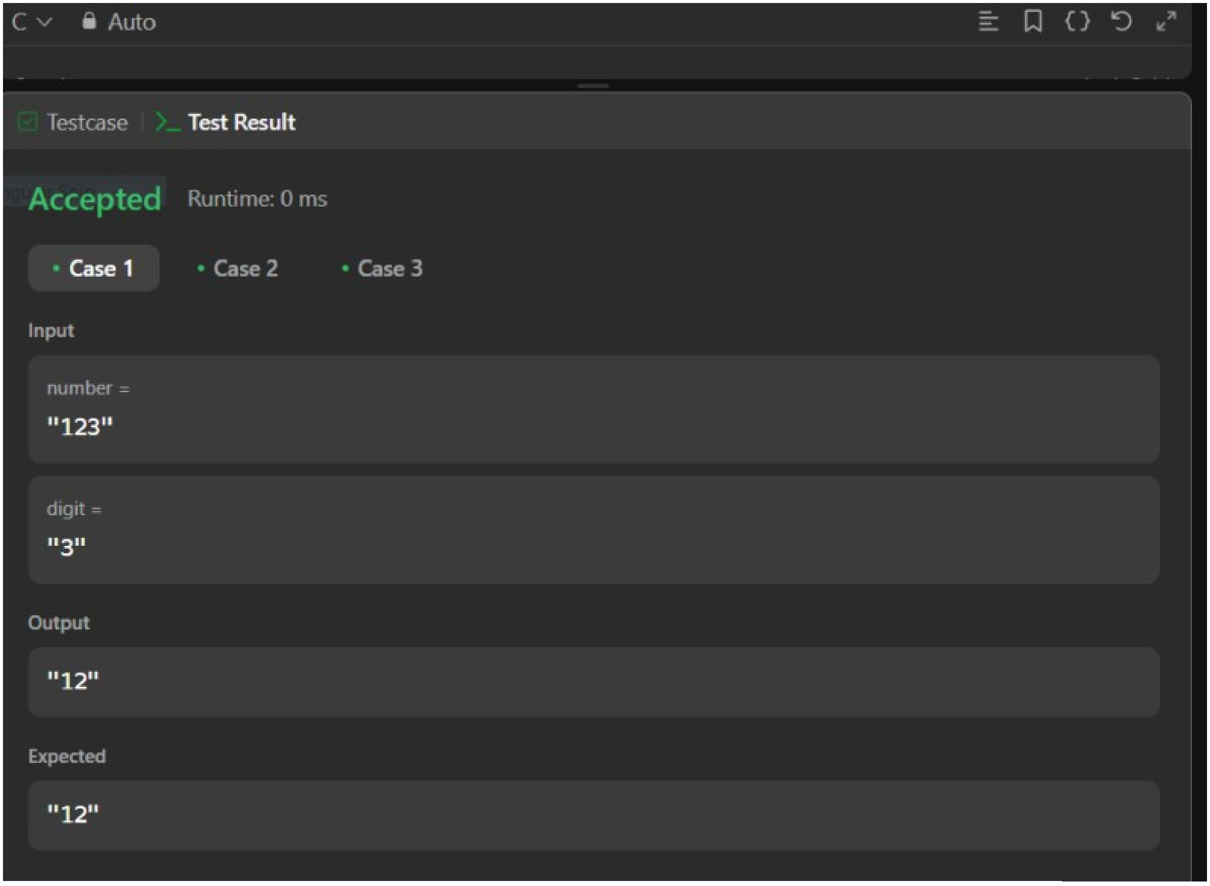
"ca"

Expected

"ca"

3b)WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & DisplayThe program should print appropriate messages for queue empty and queue overflow conditions. The program should be done using pass by reference only.

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

#define MAX 5

typedef struct {
    int data[MAX];
    int front;
    int rear;
} Queue;

void initializeQueue(Queue *q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(Queue *q) {
    return (q->rear + 1) % MAX == q->front;
}

int isEmpty(Queue *q) {
    return q->front == -1;
}

void insert(Queue *q, int value) {
    if (isFull(q)) {
        printf("Queue Overflow! Cannot insert %d.\n", value);
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear = (q->rear + 1) % MAX;
    q->data[q->rear] = value;
    printf("Inserted %d into the queue.\n", value);
}

void delete(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Queue is empty.\n");
        return;
    }
    printf("Deleted %d from the queue.\n", q->data[q->front]);
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front = (q->front + 1) % MAX;
    }
}

void display(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    int i = q->front;
    while (1) {
        printf("%d ", q->data[i]);
        if (i == q->rear) break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);
    int choice, value;

    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(&q, value);
                break;
            case 2:
                delete(&q);
                break;
            case 3:
                display(&q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
    return 0;}
```



Accepted Runtime: 0 ms

Input

number =
"123"

digit =
"3"

Output

"12"

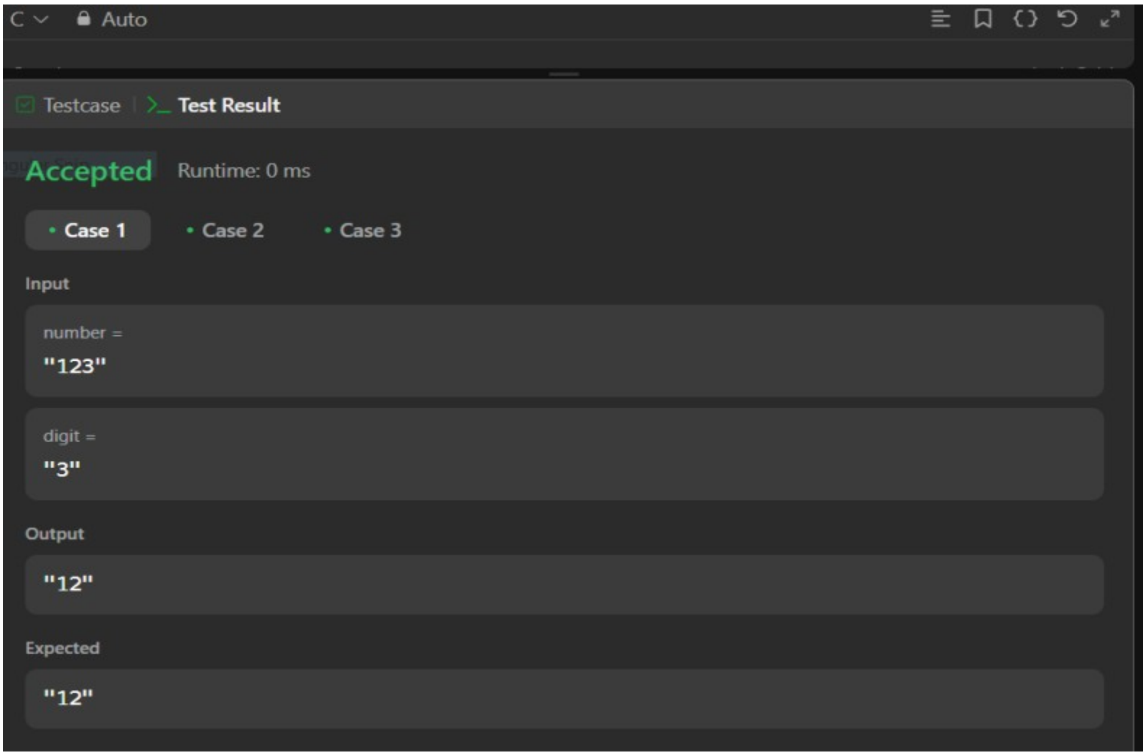
Expected

"12"

LeetCode Program- Remove Digit from Number to Maximize Result

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

```
#include <string.h>

void removeDigit(char *number, char digit, char *result) {
    int n = strlen(number), maxIndex = -1;
    for (int i = 0; i < n; i++) {
        if (number[i] == digit) {
            if (i < n - 1 && number[i] < number[i + 1]) {
                maxIndex = i;
                break;
            }
            maxIndex = i;
        }
    }
    int j = 0;
    for (int i = 0; i < n; i++) {
        if (i != maxIndex) {
            result[j++] = number[i];
        }
    }
    result[j] = '\0';
}

int main() {
    char number[100], digit, result[100];
    printf("Enter the number: ");
    scanf("%s", number);
    printf("Enter the digit to remove: ");
    scanf(" %c", &digit);
    removeDigit(number, digit, result);
    printf("Result: %s\n", result);
    return 0;
```



Accepted Runtime: 0 ms

Input

number =
"123"

digit =
"3"

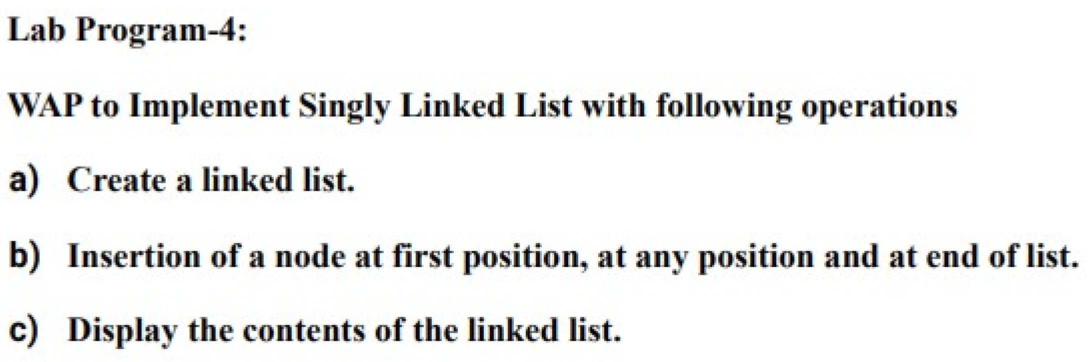
Output

"12"

Expected

"12"

```
}
```



Lab Program-4:

WAP to Implement Singly Linked List with following operations

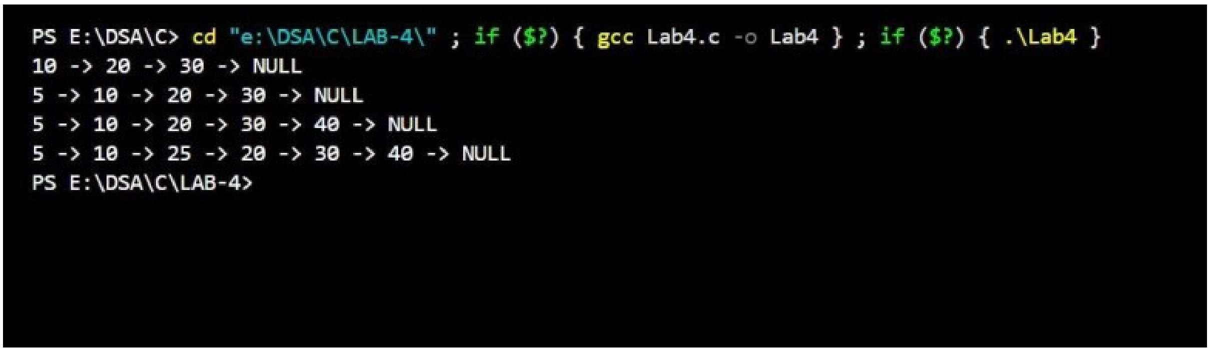
a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

c) Display the contents of the linked list.

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

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

void createLinkedList(Node **head, int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    *head = newNode;
}
```

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

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

void insertAtPosition(Node **head, int value, int position) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = value;
    if (position == 1) {
        newNode->next = *head;
        *head = newNode;
        return;
    }
    Node *temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position out of bounds.\n");
        free(newNode);
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

void displayLinkedList(Node *head) {
    if (head == NULL) {
        printf("Linked list is empty.\n");
        return;
    }
    Node *temp = head;
    printf("Linked list contents: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

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

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create Linked List\n");
        printf("2. Insert at Beginning\n");
        printf("3. Insert at End\n");
        printf("4. Insert at 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 value to create linked list: ");
                scanf("%d", &value);
                createLinkedList(&head, value);
                break;
            case 2:
                printf("Enter value to insert at beginning: ");
                scanf("%d", &value);
                insertAtBeginning(&head, value);
                break;
            case 3:
                printf("Enter value to insert at end: ");
                scanf("%d", &value);
                insertAtEnd(&head, value);
                break;
            case 4:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                printf("Enter position to insert: ");
                scanf("%d", &position);
                insertAtPosition(&head, value, position);
                break;
            case 5:
                displayLinkedList(head);
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}
```



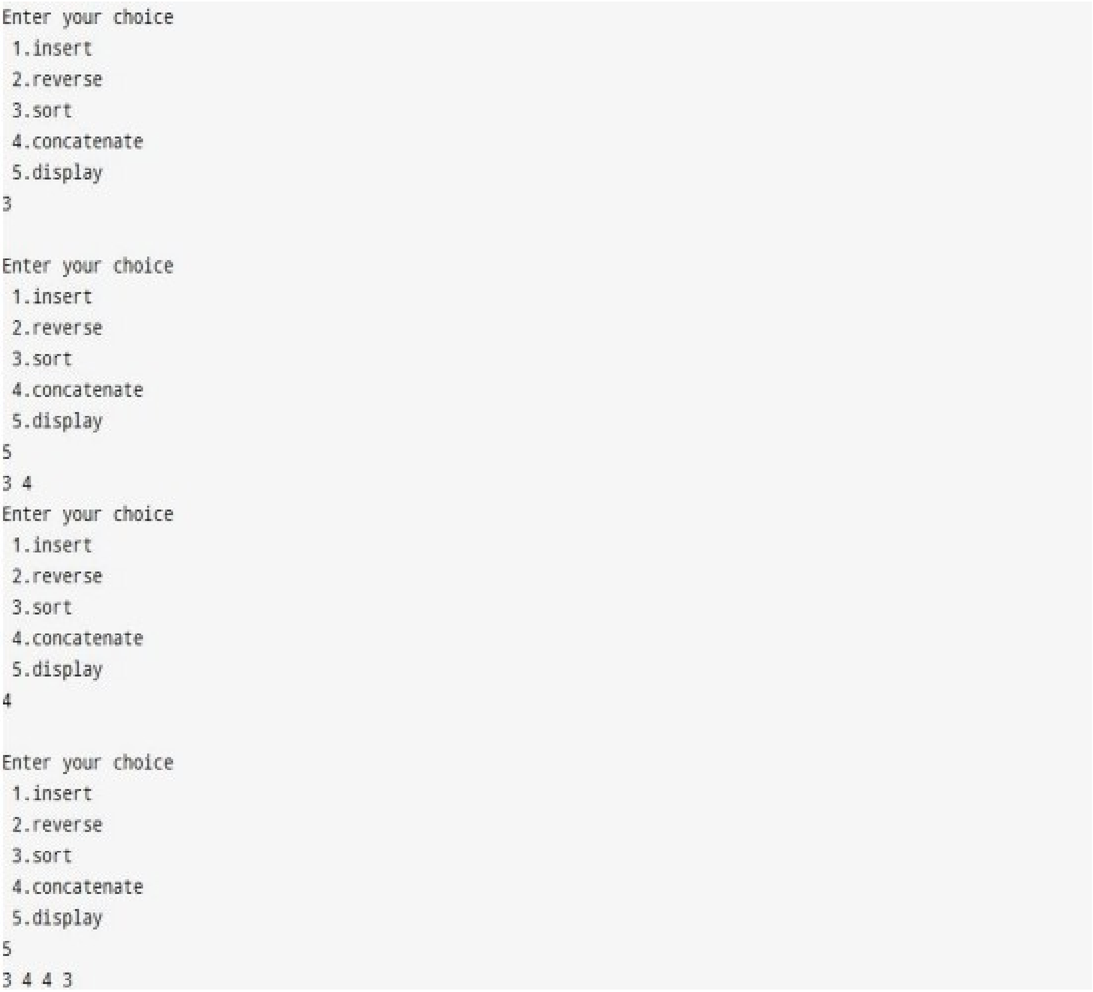
```
PS E:\DSA\C> cd "e:\DSA\C\LAB-4\" ; if ($?) { gcc Lab4.c -o Lab4 } ; if ($?) { .\Lab4 }
10 -> 20 -> 30 -> NULL
5 -> 10 -> 20 -> 30 -> NULL
5 -> 10 -> 20 -> 30 -> 40 -> NULL
5 -> 10 -> 25 -> 20 -> 30 -> 40 -> NULL
PS E:\DSA\C\LAB-4>
```

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>

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

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

void displayLinkedList(Node *head) {
    if (head == NULL) {
        printf("Linked list is empty.\n");
        return;
    }
    Node *temp = head;
    printf("Linked list contents: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

void sortLinkedList(Node **head) {
    if (*head == NULL || (*head)->next == NULL) return;
    Node *i, *j;
    int temp;
    for (i = *head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

void reverseLinkedList(Node **head) {
    Node *prev = NULL, *current = *head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void concatenateLinkedLists(Node **head1, Node **head2) {
    if (*head1 == NULL) {
        *head1 = *head2;
        return;
    }
    Node *temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = *head2;
}
```

```
int main() {
    Node *list1 = NULL, *list2 = NULL;
    int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert into List 1\n");
        printf("2. Insert into List 2\n");
        printf("3. Display List 1\n");
        printf("4. Display List 2\n");
        printf("5. Sort List 1\n");
        printf("6. Reverse List 1\n");
        printf("7. Concatenate List 2 into List 1\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert into List 1: ");
                scanf("%d", &value);
                insertAtEnd(&list1, value);
                break;
            case 2:
                printf("Enter value to insert into List 2: ");
                scanf("%d", &value);
                insertAtEnd(&list2, value);
                break;
            case 3:
                printf("List 1: ");
                displayLinkedList(list1);
                break;
            case 4:
                printf("List 2: ");
                displayLinkedList(list2);
                break;
            case 5:
                sortLinkedList(&list1);
                printf("List 1 sorted.\n");
                break;
            case 6:
                reverseLinkedList(&list1);
                printf("List 1 reversed.\n");
                break;
            case 7:
                concatenateLinkedLists(&list1, &list2);
                printf("List 2 concatenated into List 1.\n");
                break;
            case 8:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

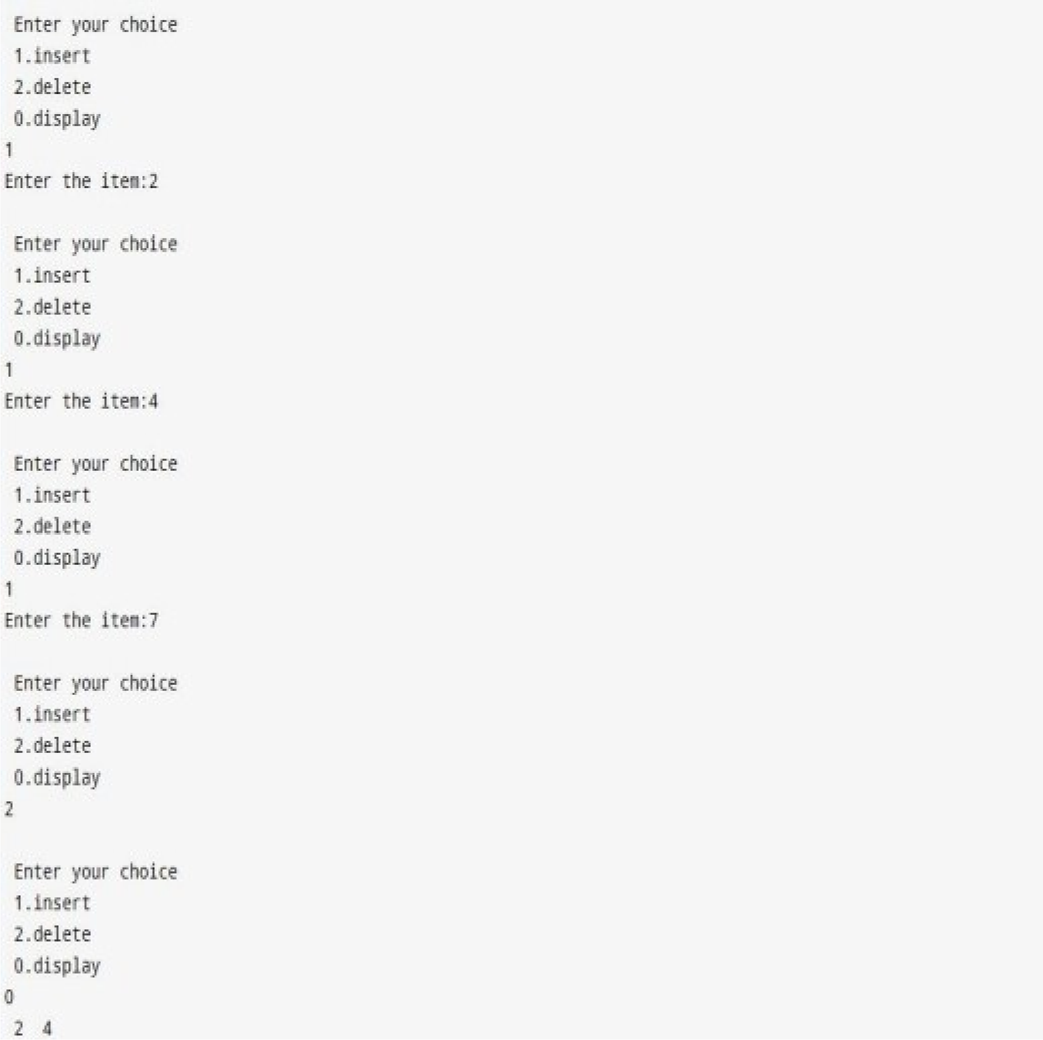


```
Enter your choice
 1.insert
 2.reverse
 3.sort
 4.concatenate
 5.display
3

Enter your choice
 1.insert
 2.reverse
 3.sort
 4.concatenate
 5.display
5
3 4
Enter your choice
 1.insert
 2.reverse
 3.sort
 4.concatenate
 5.display
4

Enter your choice
 1.insert
 2.reverse
 3.sort
 4.concatenate
 5.display
5
3 4 4 3
```

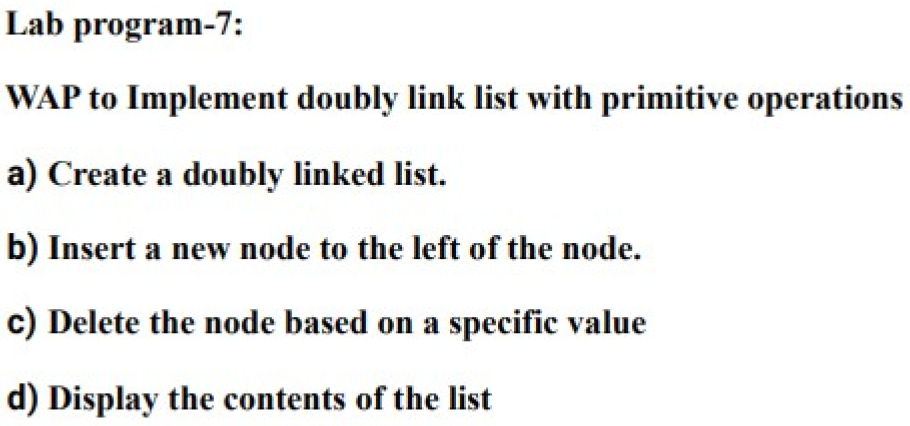
6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

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

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

// Stack Operations
```

```
void push(Node** top, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = *top;
    *top = newNode;
}

int pop(Node** top) {
    if (*top == NULL) {
        printf("Stack Underflow\n");
        return -1;
    }
    Node* temp = *top;
    int poppedValue = temp->data;
    *top = temp->next;
    free(temp);
    return poppedValue;
}

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

// Queue Operations

void enqueue(Node** front, Node** rear, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    if (*rear == NULL) {
        *front = *rear = newNode;
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
}

int dequeue(Node** front, Node** rear) {
    if (*front == NULL) {
        printf("Queue Underflow\n");
        return -1;
    }
    Node* temp = *front;
    int dequeuedValue = temp->data;
    *front = (*front)->next;
    if (*front == NULL) {
        *rear = NULL;
    }
    free(temp);
    return dequeuedValue;
}

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

int main() {
    Node* stackTop = NULL;
    Node* queueFront = NULL;
    Node* queueRear = NULL;
    int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Push to Stack\n");
        printf("2. Pop from Stack\n");
        printf("3. Display Stack\n");
        printf("4. Enqueue to Queue\n");
        printf("5. Dequeue from Queue\n");
        printf("6. Display Queue\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push to stack: ");
                scanf("%d", &value);
                push(&stackTop, value);
                break;
            case 2:
                value = pop(&stackTop);
                if (value != -1) {
                    printf("Popped value: %d\n", value);
                }
                break;
            case 3:
                displayStack(stackTop);
                break;
            case 4:
                printf("Enter value to enqueue to queue: ");
                scanf("%d", &value);
                enqueue(&queueFront, &queueRear, value);
                break;
            case 5:
                value = dequeue(&queueFront, &queueRear);
                if (value != -1) {
                    printf("Dequeued value: %d\n", value);
                }
                break;
            case 6:
                displayQueue(queueFront);
                break;
            case 7:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}
```



```
 Enter your choice
 1.insert
 2.delete
 0.display
1
Enter the item:2

 Enter your choice
 1.insert
 2.delete
 0.display
1
Enter the item:4

 Enter your choice
 1.insert
 2.delete
 0.display
1
Enter the item:7

 Enter your choice
 1.insert
 2.delete
 0.display
2

 Enter your choice
 1.insert
 2.delete
 0.display
0
 2  4
```



Lab program-7:

WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.

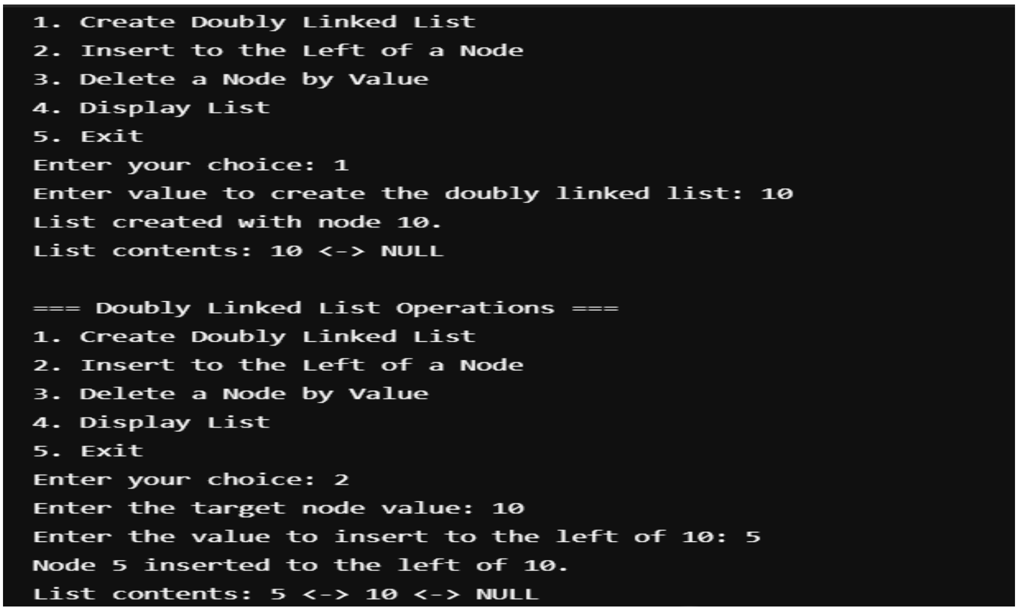
b) Insert a new node to the left of the node.

c) Delete the node based on a specific value

d) Display the contents of the list

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

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

```
void createDoublyLinkedList(Node** head, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;
    *head = newNode;
}

void insertToLeft(Node** head, int target, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;

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

    if (current == NULL) {
        printf("Target node not found.\n");
        free(newNode);
        return;
    }

    newNode->next = current;
    newNode->prev = current->prev;

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

    current->prev = newNode;
}

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

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

    if (current == NULL) {
        printf("Node with value %d not found.\n", value);
        return;
    }

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

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

    free(current);
}

void displayList(Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }

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

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

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create Doubly Linked List\n");
        printf("2. Insert to the Left of a Node\n");
        printf("3. Delete a Node by Value\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to create the doubly linked list: ");
                scanf("%d", &value);
                createDoublyLinkedList(&head, value);
                break;
            case 2:
                printf("Enter the target node value: ");
                scanf("%d", &target);
                printf("Enter the value to insert to the left of %d: ", target);
                scanf("%d", &value);
                insertToLeft(&head, target, value);
                break;
            case 3:
                printf("Enter the value of the node to delete: ");
                scanf("%d", &value);
                deleteNode(&head, value);
                break;
            case 4:
                displayList(head);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```



```
1. Create Doubly Linked List
2. Insert to the Left of a Node
3. Delete a Node by Value
4. Display List
5. Exit
Enter your choice: 1
Enter value to create the doubly linked list: 10
List created with node 10.
List contents: 10 <-> NULL

=== Doubly Linked List Operations ===
1. Create Doubly Linked List
2. Insert to the Left of a Node
3. Delete a Node by Value
4. Display List
5. Exit
Enter your choice: 2
Enter the target node value: 10
Enter the value to insert to the left of 10: 5
Node 5 inserted to the left of 10.
List contents: 5 <-> 10 <-> NULL
```

Lab program-8: Write

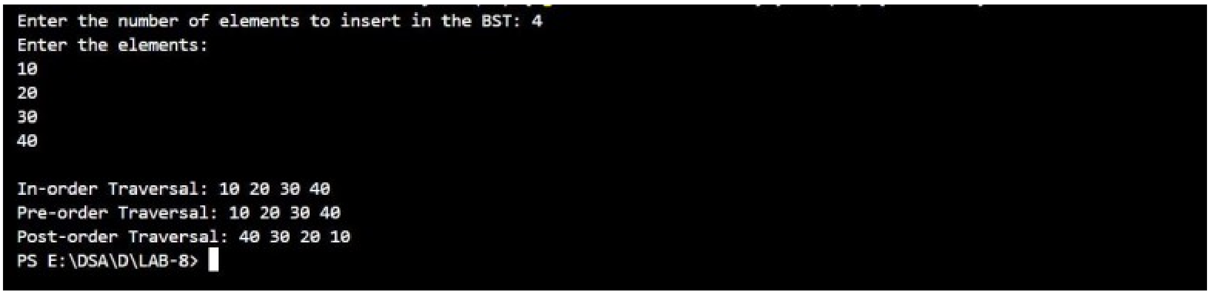
a program

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.

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

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

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

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

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

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

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

void displayTree(Node* root) {
    if (root == NULL) {
        printf("Tree is empty.\n");
        return;
    }

    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Pre-order Traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Post-order Traversal: ");
    postorderTraversal(root);
    printf("\n");
}

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

    while (1) {
        printf("\n=== Binary Search Tree Operations ===\n");
        printf("1. Insert into BST\n");
        printf("2. Traverse the Tree (In-order, Pre-order, Post-order)\n");
        printf("3. Display the Tree\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert into the BST: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("Tree Traversals:\n");
                displayTree(root);
                break;
            case 3:
                displayTree(root);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```

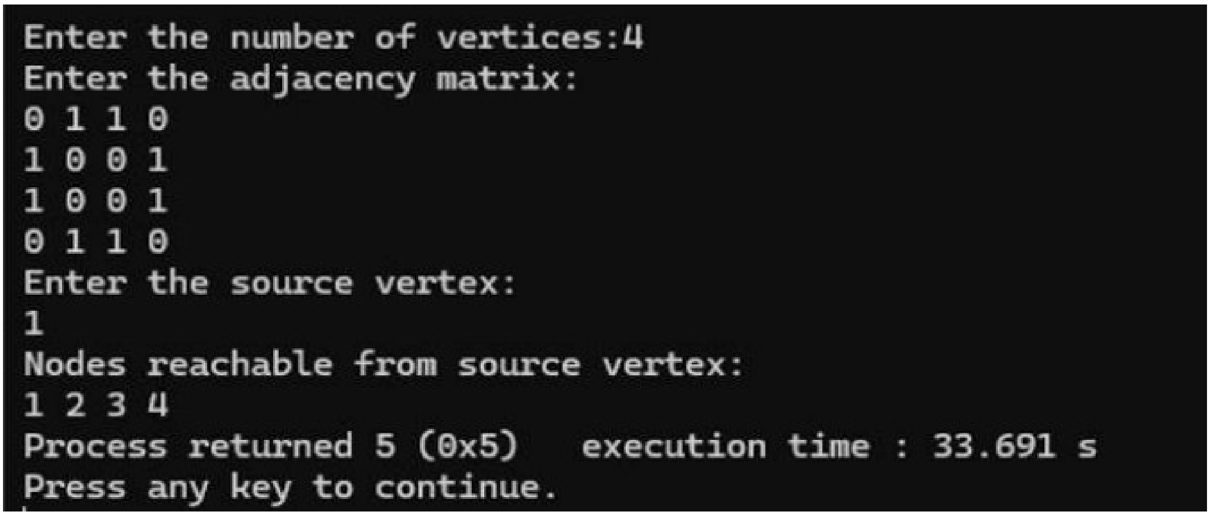


```
Enter the number of elements to insert in the BST: 4
Enter the elements:
10
20
30
40

In-order Traversal: 10 20 30 40
Pre-order Traversal: 10 20 30 40
Post-order Traversal: 40 30 20 10
PS E:\DSA\D\LAB-8> 
```

LAB PROGRAM 9-

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

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

#define MAX 100

int adj[MAX][MAX]; // Adjacency matrix
int visited[MAX];  // Visited array
int queue[MAX], front = -1, rear = -1;

void enqueue(int vertex) {
    if (rear == MAX - 1) {
        printf("Queue overflow\n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    queue[++rear] = vertex;
}

int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue underflow\n");
        return -1;
    }
    return queue[front++];
}

int isQueueEmpty() {
    return (front == -1 || front > rear);
}

void bfs(int startVertex, int n) {
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }

    enqueue(startVertex);
    visited[startVertex] = 1;

    printf("BFS Traversal: ");
    while (!isQueueEmpty()) {
        int currentVertex = dequeue();
        printf("%d ", currentVertex);

        for (int i = 0; i < n; i++) {
            if (adj[currentVertex][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}

int main() {
    int n, e, u, v, startVertex;

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

    printf("Enter the number of edges in the graph: ");
    scanf("%d", &e);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            adj[i][j] = 0;
        }
    }

    printf("Enter the edges (u v):\n");
    for (int i = 0; i < e; i++) {
        scanf("%d %d", &u, &v);
        adj[u][v] = 1;
        adj[v][u] = 1; // For undirected graph
    }

    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);

    bfs(startVertex, n);

    return 0;
}
```

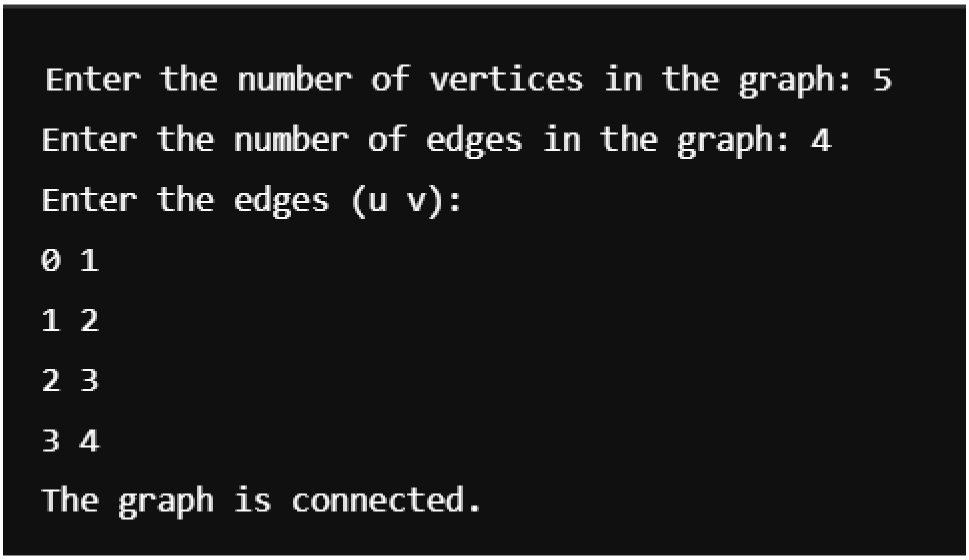


```
Enter the number of vertices:4
Enter the adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the source vertex:
1
Nodes reachable from source vertex:
1 2 3 4
Process returned 5 (0x5)   execution time : 33.691 s
Press any key to continue.
```

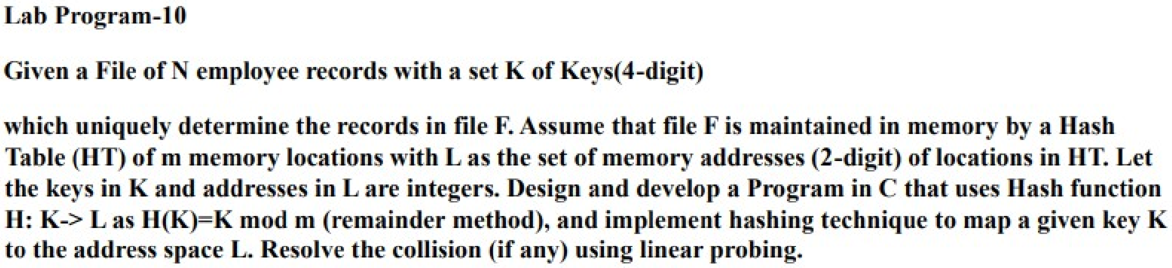
9b) Write a program to check whether given graph is connected or not using DFS method.

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

#define MAX 100

int adj[MAX][MAX];
int visited[MAX];

void dfs(int vertex, int n) {
    visited[vertex] = 1;
    for (int i = 0; i < n; i++) {
        if (adj[vertex][i] == 1 && !visited[i]) {
            dfs(i, n);
        }
    }
}
```

```
int isConnected(int n) {
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    dfs(0, n);
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            return 0;
        }
    }
    return 1;
}

int main() {
    int n, e, u, v;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &n);
    printf("Enter the number of edges in the graph: ");
    scanf("%d", &e);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            adj[i][j] = 0;
        }
    }
    printf("Enter the edges (u v):\n");
    for (int i = 0; i < e; i++) {
        scanf("%d %d", &u, &v);
        adj[u][v] = 1;
        adj[v][u] = 1;
    }
    if (isConnected(n)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }
    return 0;
}
```



```
Enter the number of vertices in the graph: 5
Enter the number of edges in the graph: 4
Enter the edges (u v):
0 1
1 2
2 3
3 4
The graph is connected.
```



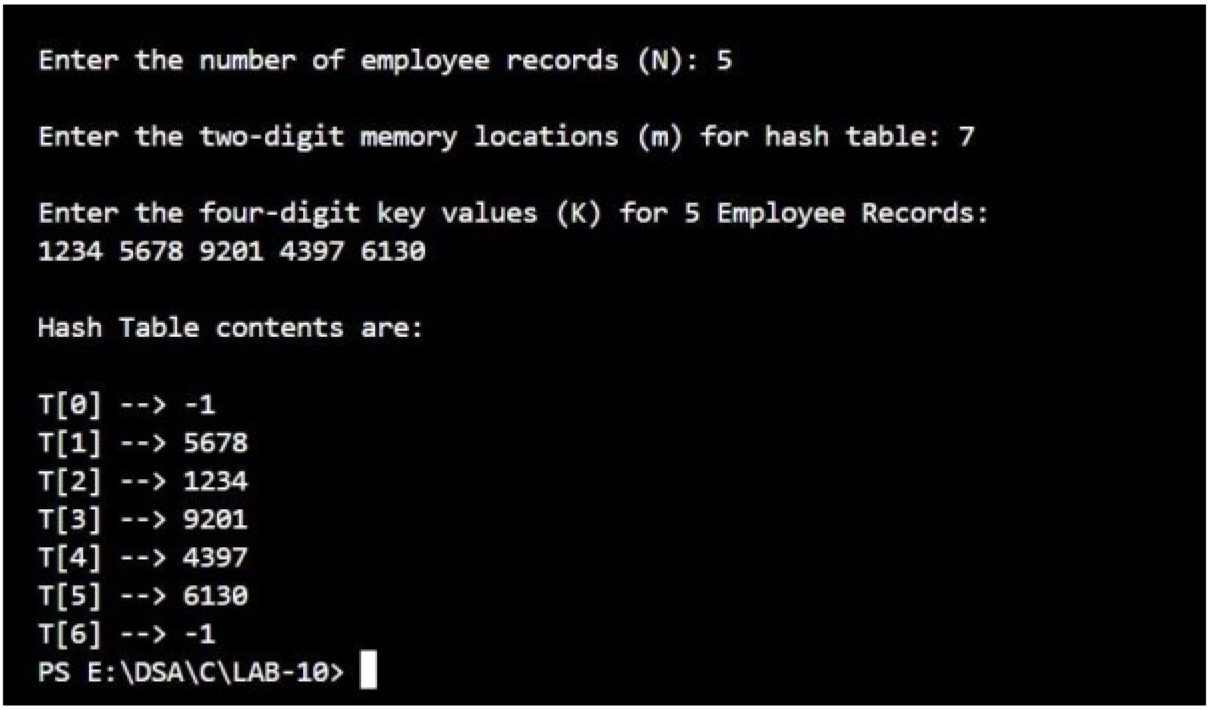
Lab Program-10

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>
#include <stdlib.h>

#define MAX 100

int hashTable[MAX];

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

int hashFunction(int key, int m) {
    return key % m;
}

void insert(int key, int m) {
    int index = hashFunction(key, m);
    int originalIndex = index;
    while (hashTable[index] != -1) {
        index = (index + 1) % m;
        if (index == originalIndex) {
            printf("Hash table is full. Cannot insert key: %d\n", key);
            return;
        }
    }
    hashTable[index] = key;
    printf("Key %d inserted at index %d\n", key, index);
}

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

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

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

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

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

    displayHashTable(m);
    return 0;
```

```
}
```



```
Enter the number of employee records (N): 5

Enter the two-digit memory locations (m) for hash table: 7

Enter the four-digit key values (K) for 5 Employee Records:
1234 5678 9201 4397 6130

Hash Table contents are:

T[0] --> -1
T[1] --> 5678
T[2] --> 1234
T[3] --> 9201
T[4] --> 4397
T[5] --> 6130
T[6] --> -1
PS E:\DSA\C\LAB-10>
```