VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

RAJA VISHWANATH DASARI (1BM23CS259)

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 RAJA VISHWANATH DASARI **(1BM23CS259)**, 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 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.

Prof. Lakshmi Neelima M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

INDEX

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.

Program:

```
#include <stdio.h>
#define SIZE 3

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

void push(int val) {
    if (top == SIZE - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = val;
    printf("%d pushed into the stack\n", val);
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return;
    }
```

```
    printf("%d popped from the stack\n", stack[top--]);
}

void display() {
    if (top == -1) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements: ");
    for (int i = top; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}

int main() {
    int choice, val;

    while (1) {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

Output:

```
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 12
12 pushed into the stack
Enter your choice: 1
Enter value to push: 13
13 pushed into the stack
Enter your choice: 1
Enter value to push: 14
14 pushed into the stack
Enter your choice: 1
Enter value to push: 123
Stack Overflow
Enter your choice: 3
Stack elements: 14 13 12
Enter your choice: 2
14 popped from the stack
Enter your choice: 2
13 popped from the stack
Enter your choice: 2
12 popped from the stack
Enter your choice: 2
Stack Underflow
Enter your choice: 3
Stack is empty
Enter your choice: 4

Process returned 0 (0x0)   execution time : 27.636 s
Press any key to continue.
```

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)

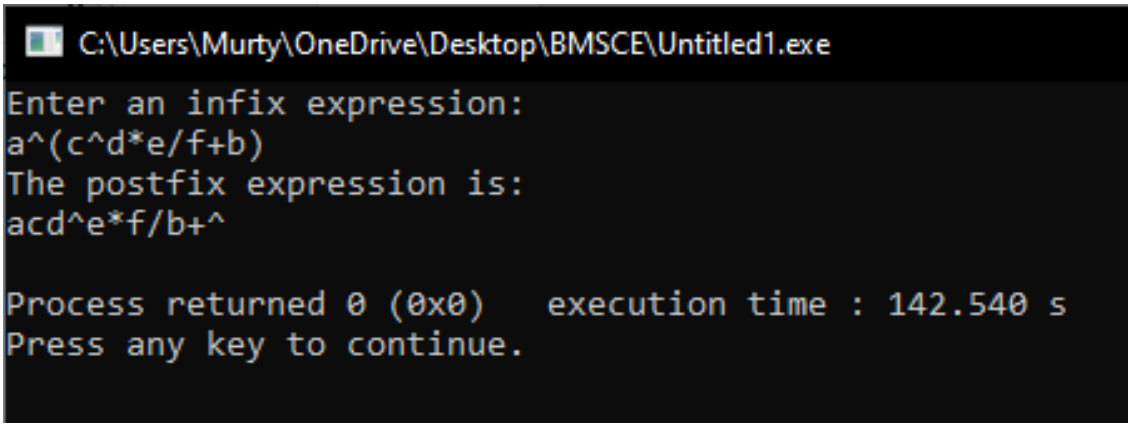
Program:

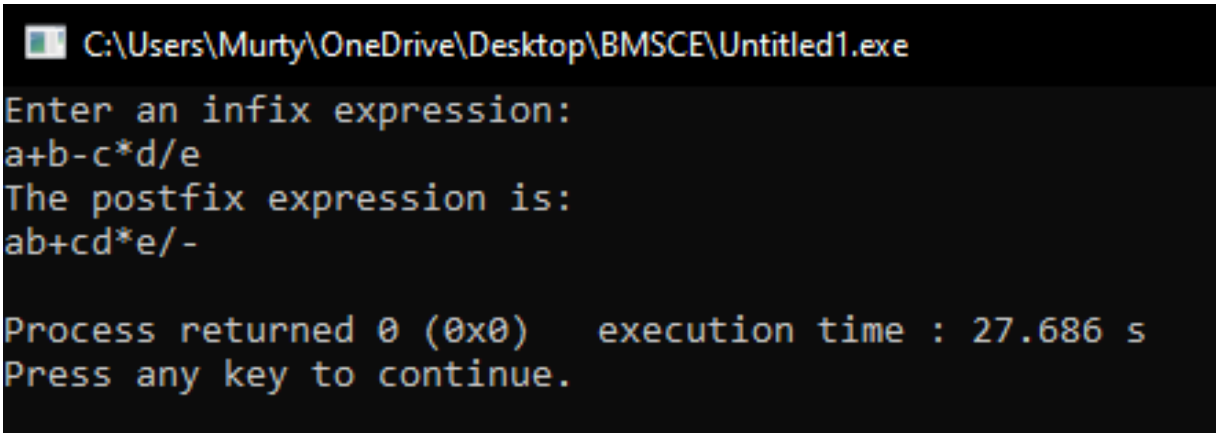
```
#include <stdio.h>

#define size 100


void push(char x);

char pop();

int precedence(char x);

int isOperand(char x);

void infixTopostfix(char* exp);


char stack[size];

int top = -1;


void push(char x) {
    if (top == size - 1) {
        printf("Stack Overflow\n");
    } else {
        stack[++top] = x;
    }
}


char pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;  // Return an invalid character on underflow
    } else {
```

```
        char temp = stack[top];
        top--;
        return temp;
    }
}

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

int isOperand(char x) {
    // Check if x is an operand (A-Z, a-z, 0-9)
    return (x >= 'A' && x <= 'Z') || (x >= 'a' && x <= 'z') || (x >= '0' && x <= '9');
}

void infixTopostfix(char* exp) {
    char postfix[size];
    int i = 0, j = 0;

    for (i = 0; exp[i] != '\0'; i++) {
        char ch = exp[i];
        if (isOperand(ch)) {
            postfix[j++] = ch;
        } else if (ch == '(') {
            push(ch);
        } else if (ch == ')') {
```

```
            while (top != -1 && stack[top] != '(') {
                postfix[j++] = pop();
            }
            pop();  // Discard the '('
        } else {  // Operator case
            while (top != -1 && precedence(stack[top]) >= precedence(ch)) {
                postfix[j++] = pop();
            }
            push(ch);
        }
    }

    // Pop all the remaining operators from the stack
    while (top != -1) {
        postfix[j++] = pop();
    }

    postfix[j] = '\0';  // Null-terminate the postfix string
    puts(postfix);      // Print the result
}
int main() {
    printf("Enter an infix expression:\n");
    char infix[size];
    scanf("%s", infix);
    printf("The postfix expression is:\n");
    infixTopostfix(infix);
    return 0;

}
```

Output:


C:\Users\Murty\OneDrive\Desktop\BMSCE\Untitled1.exe
Enter an infix expression:
a^(c^d*e/f+b)
The postfix expression is:
acd^e*f/b+^
Process returned 0 (0x0)   execution time : 142.540 s
Press any key to continue.


C:\Users\Murty\OneDrive\Desktop\BMSCE\Untitled1.exe
Enter an infix expression:
a+b-c*d/e
The postfix expression is:
ab+cd*e/-
Process returned 0 (0x0)   execution time : 27.686 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.

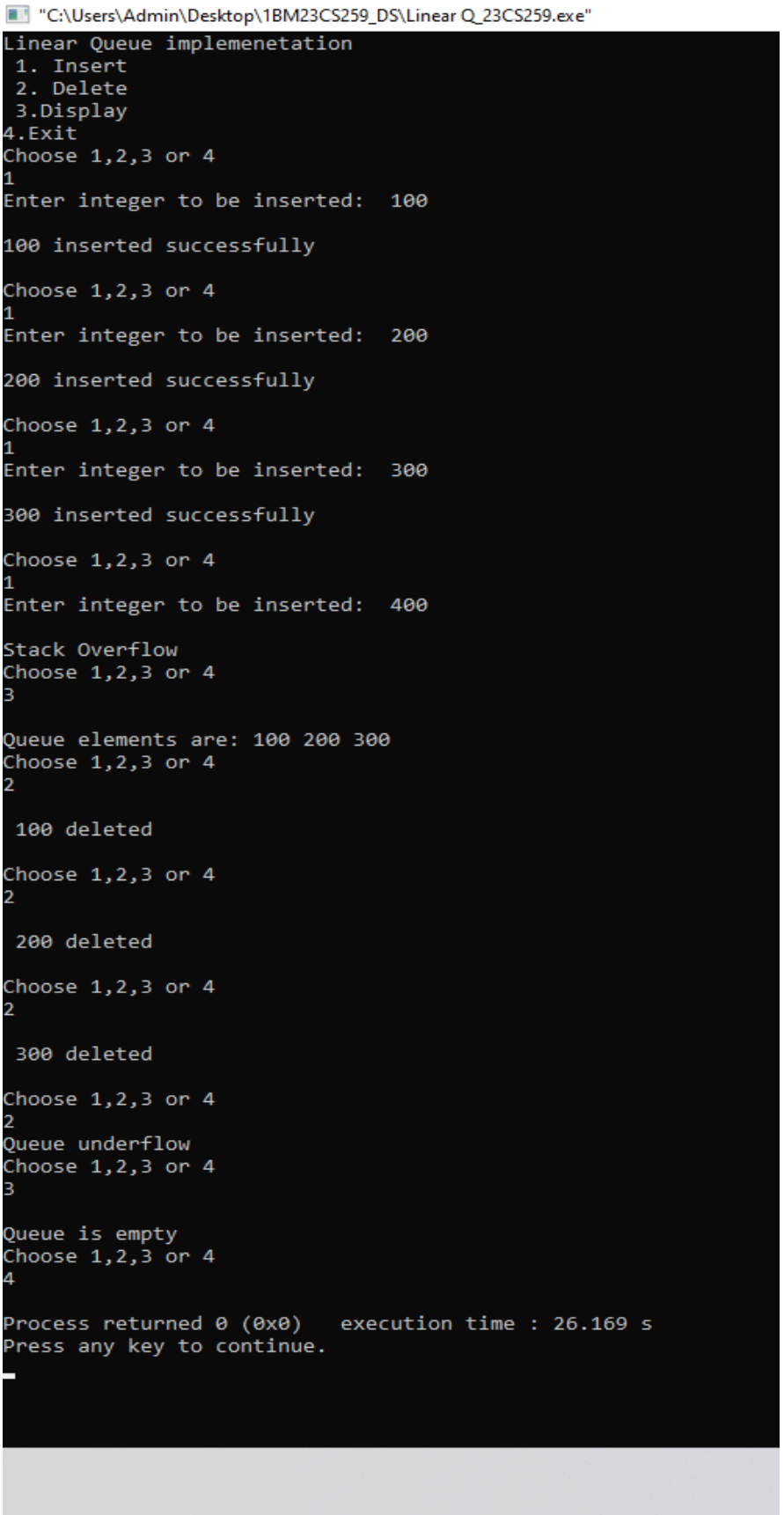
Program:

```
#include<stdio.h>

#define size 3


int queue[size];

int front = -1;

int rear = -1;


void delete();

void insert(int x);

void display();


void insert(int n){
    if(rear == size -1){
        printf("\nStack Overflow\n");
        return;
    }
    if(front == -1){
        front=0;
    }
    rear++;
    queue[rear] = n;
    printf("\n%d inserted successfully\n\n",n);
}
```

```
void delete(){
    if(front == -1 || front>rear){
        printf("Queue underflow\n");
        return;
    }
    printf("\n %d deleted\n\n",queue[front]);
    front++;
    if(front>rear){
        front = 1;
        rear = -1;
    }

}

void display(){
    if(front == -1 || front>rear){
        printf("\nQueue is empty \n");
        return;
    }
    printf("\nQueue elements are: ");
    for(int i = front; i<=rear; i++){
        printf("%d ",queue[i]);
    }
    printf("\n");

}
int main(){
    int option, elem;
    int flag = 0;
```

```
    printf("Linear Queue implemenetation\n 1. Insert\n 2. Delete\n
3.Display\n4.Exit\n");
    while(flag==0){
        printf("Choose 1,2,3 or 4\n");
        scanf("%d",&option);
        switch(option){
            case 1: printf("Enter integer to be inserted:  ");
                    scanf("%d",&elem);
                    insert(elem);
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
            case 4: flag = 1;
                    break;
            default: printf("Enter a valid option\n");
        }
    }
    return 0;

}
```

Output:


"C:\Users\Admin\Desktop\1BM23CS259_DS\Linear Q_23CS259.exe"
Linear Queue implemenetation
1. Insert
2. Delete
3.Display
4.Exit
Choose 1,2,3 or 4
1
Enter integer to be inserted: 100
100 inserted successfully
Choose 1,2,3 or 4
1
Enter integer to be inserted: 200
200 inserted successfully
Choose 1,2,3 or 4
1
Enter integer to be inserted: 300
300 inserted successfully
Choose 1,2,3 or 4
1
Enter integer to be inserted: 400
Stack Overflow
Choose 1,2,3 or 4
3
Queue elements are: 100 200 300
Choose 1,2,3 or 4
2
100 deleted
Choose 1,2,3 or 4
2
200 deleted
Choose 1,2,3 or 4
2
300 deleted
Choose 1,2,3 or 4
2
Queue underflow
Choose 1,2,3 or 4
3
Queue is empty
Choose 1,2,3 or 4
4
Process returned 0 (0x0) execution time : 26.169 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.

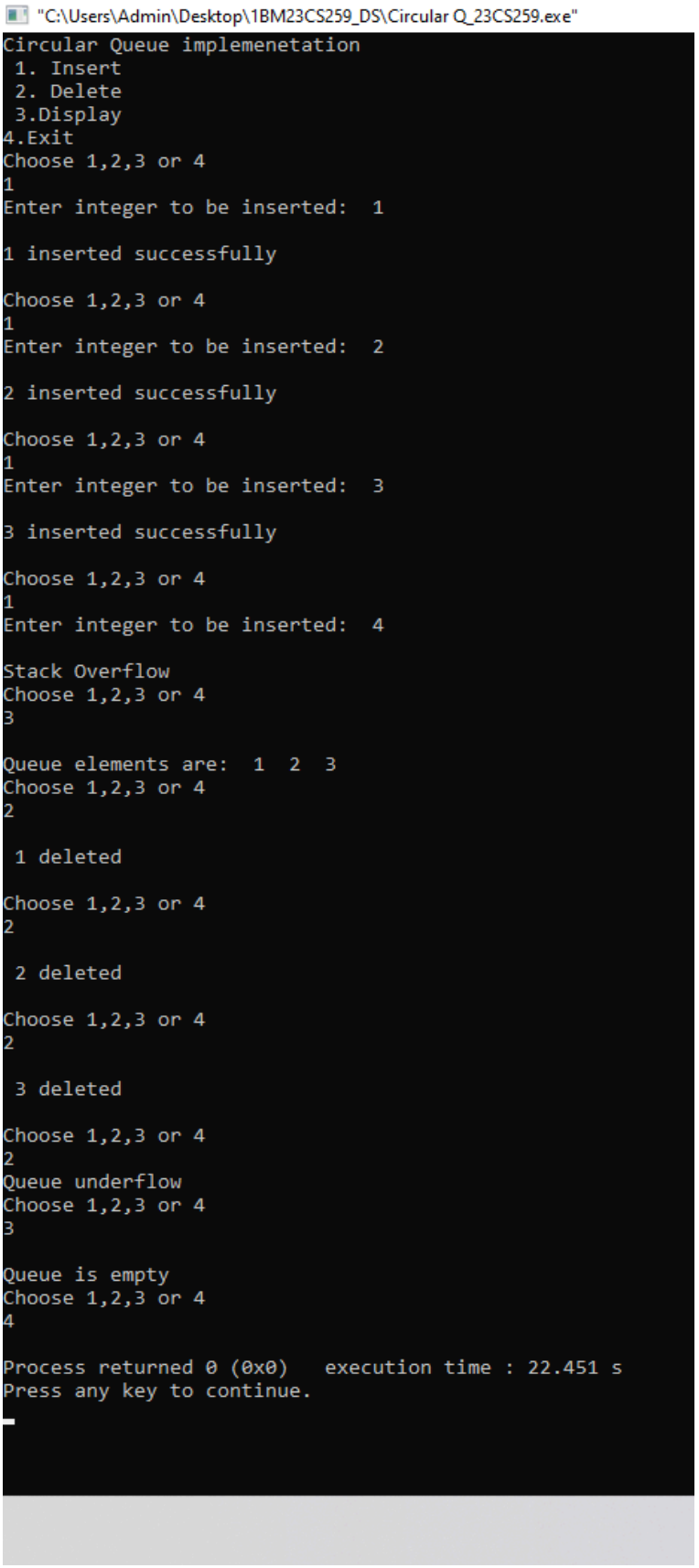
Program:

```
#include<stdio.h>

#define size 3


int cqueue[size];

int front = -1;

int rear = -1;


void delete();

void insert(int x);

void display();


void insert(int n){
    if((rear+1)%size == front){
        printf("\nStack Overflow\n");
        return;
    }
    if(front == -1){
        front=0;
    }
    rear= (rear+1)%size;
    cqueue[rear] = n;
    printf("\n%d inserted successfully\n\n",n);
}


void delete() {
```

```
    if (front == -1) {
        printf("Queue underflow\n");
        return;
    }
    printf("\n %d deleted\n\n", cqueue[front]);
    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % size;
    }
}

void display(){
    if(front == -1 || front>rear){
        printf("\nQueue is empty \n");
        return;
    }
    printf("\nQueue elements are: ");
    int i = front;
    while(i!=rear){
        printf(" %d ",cqueue[i]);
        i = (i+1)%size;
    }
    printf(" %d",cqueue[i]);
    printf("\n");

}
int main(){
```

```
    int option, elem;
    int flag = 0;
    printf("Circular Queue implemenetation\n 1. Insert\n 2. Delete\n
3.Display\n4.Exit\n");
    while(flag==0){
        printf("Choose 1,2,3 or 4\n");
        scanf("%d",&option);
        switch(option){
            case 1: printf("Enter integer to be inserted:  ");
                    scanf("%d",&elem);
                    insert(elem);
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
            case 4: flag = 1;
                    break;
            default: printf("Enter a valid option\n");
        }
    }
    return 0;

}
```

Output:


"C:\Users\Admin\Desktop\1BM23CS259_DS\Circular Q_23CS259.exe"
Circular Queue implemenetation
1. Insert
2. Delete
3.Display
4.Exit
Choose 1,2,3 or 4
1
Enter integer to be inserted: 1
1 inserted successfully
Choose 1,2,3 or 4
1
Enter integer to be inserted: 2
2 inserted successfully
Choose 1,2,3 or 4
1
Enter integer to be inserted: 3
3 inserted successfully
Choose 1,2,3 or 4
1
Enter integer to be inserted: 4
Stack Overflow
Choose 1,2,3 or 4
3
Queue elements are: 1 2 3
Choose 1,2,3 or 4
2
1 deleted
Choose 1,2,3 or 4
2
2 deleted
Choose 1,2,3 or 4
2
3 deleted
Choose 1,2,3 or 4
2
Queue underflow
Choose 1,2,3 or 4
3
Queue is empty
Choose 1,2,3 or 4
4
Process returned 0 (0x0) execution time : 22.451 s
Press any key to continue.

LAB PROGRAM 4 and LAB PROGRAM 5

Both lab programs are combined in the same program.

4) WAP to Implement Singly Linked List with following operations

a) Createalinkedlist.

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

c) Display the contents of the linked list.

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.

Program:

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

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

struct node *start = NULL;

struct node *create_ll(struct node *start) {
   int value;
   struct node *newnode, *ptr;

   printf("Enter values for the nodes. Enter -1 to stop.\n");

   while (1) {
      printf("Enter value: ");
```

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

        if (value == -1) {
            break;
        }

        newnode = (struct node*)malloc(sizeof(struct node));
        newnode->data = value;
        newnode->next = NULL;

        if (start == NULL) {
            start = newnode;
        } else {
            ptr = start;
            while (ptr->next != NULL) {
                ptr = ptr->next;
            }
            ptr->next = newnode;
        }
    }

    return start;
}

struct node *insert_beg(struct node *start) {
    int val;
    struct node *newnode;
    printf("Enter the value to be inserted at the beginning:\n");
    scanf("%d", &val);
```

```
    newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    newnode->next = start;
    start = newnode;
    return start;
}

struct node *insert_last(struct node *start) {
    int num;
    struct node *newnode, *ptr;
    printf("Enter the value to be inserted at last:\n");
    scanf("%d", &num);
    newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = num;
    newnode->next = NULL;

    if (start == NULL) {
        start = newnode;
    } else {
        ptr = start;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        ptr->next = newnode;
    }

    return start;
}
```

```
struct node *insert_pos(struct node *start) {
    int num, pos;
    struct node *newnode;
    printf("Enter value to be inserted:\n");
    scanf("%d", &num);
    printf("Enter position to be inserted:\n");
    scanf("%d", &pos);

    newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = num;

    if (pos == 1) {
        start = insert_beg(start);
        return start;
    }

    struct node *ptr = start;
    for (int i = 1; i < pos - 1; i++) {
        if (ptr == NULL) {
            printf("Position out of range.\n");
            return start;
        }
        ptr = ptr->next;
    }

    newnode->next = ptr->next;
    ptr->next = newnode;

    return start;
```

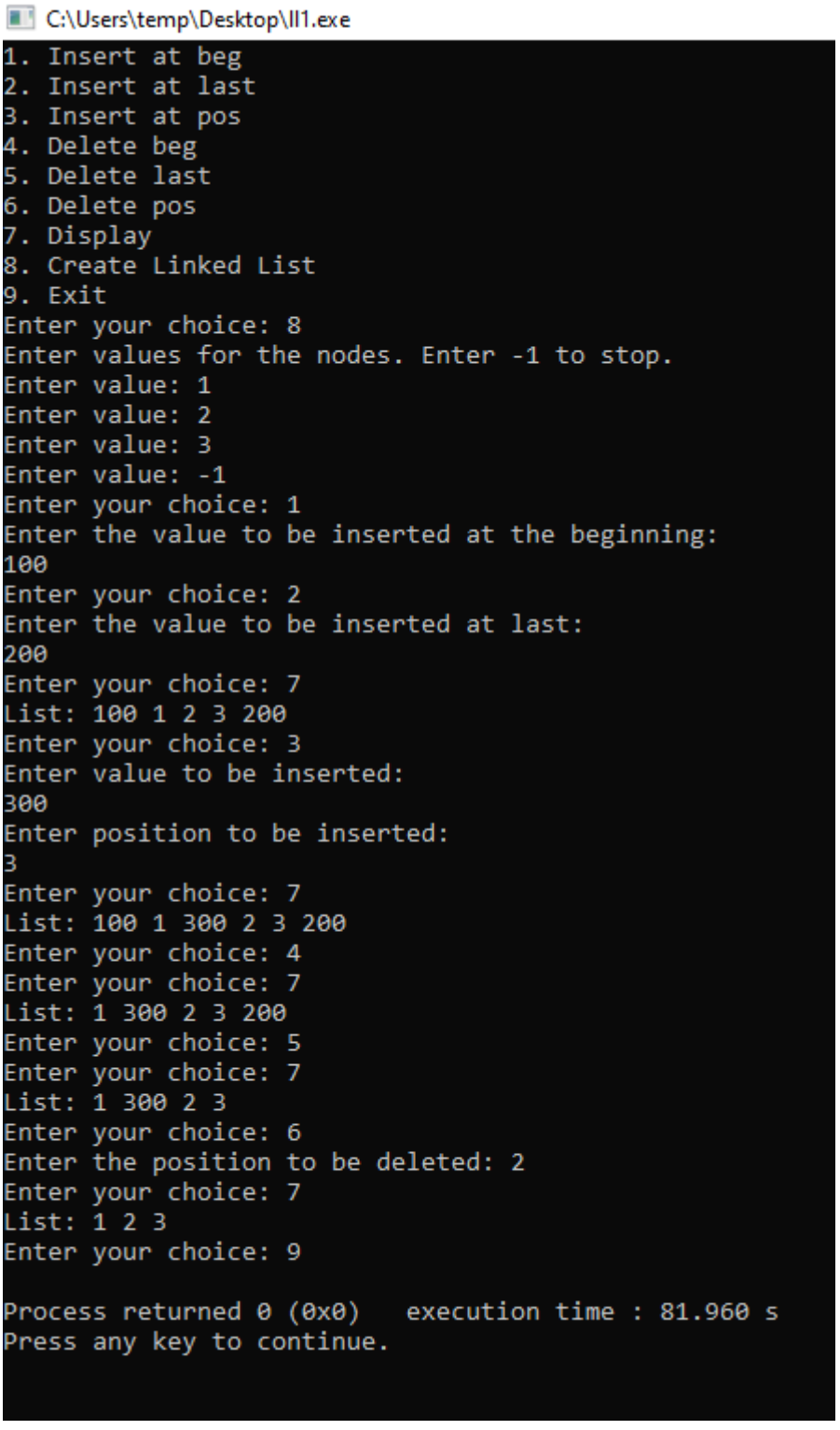
```
}

struct node *display(struct node *start) {
    struct node *ptr;
    ptr = start;

    if (ptr == NULL) {
        printf("List is empty.\n");
        return start;
    }

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

    return start;
}

struct node *delete_beg(struct node *start) {
    if (start == NULL) {
        printf("Empty list, nothing to delete.\n");
        return start;
    }

    struct node *ptr = start;
    start = start->next;
```

```
    free(ptr);
    return start;
}

struct node *delete_last(struct node *start) {
    if (start == NULL) {
        printf("Empty list, nothing to delete.\n");
        return start;
    }

    struct node *ptr = start, *preptr = NULL;
    while (ptr->next != NULL) {
        preptr = ptr;
        ptr = ptr->next;
    }

    if (preptr == NULL) {
        free(ptr);
        return NULL;
    }

    preptr->next = NULL;
    free(ptr);
    return start;
}

struct node *delete_pos(struct node *start) {
    int pos;
    printf("Enter the position to be deleted: ");
```

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

    if (pos == 1) {
        start = delete_beg(start);
        return start;
    }

    struct node *ptr = start, *preptr = NULL;
    for (int i = 1; i < pos; i++) {
        if (ptr == NULL) {
            printf("Position out of range.\n");
            return start;
        }
        preptr = ptr;
        ptr = ptr->next;
    }

    preptr->next = ptr->next;
    free(ptr);
    return start;
}

int main() {
    int choice;
    printf("1. Insert at beg\n2. Insert at last\n3. Insert at pos\n4. Delete beg\n5. Delete last\n6. Delete pos\n7. Display\n8. Create Linked List\n9. Exit\n");

    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1: start = insert_beg(start); break;
            case 2: start = insert_last(start); break;
            case 3: start = insert_pos(start); break;
            case 4: start = delete_beg(start); break;
            case 5: start = delete_last(start); break;
            case 6: start = delete_pos(start); break;
            case 7: start = display(start); break;
            case 8: start = create_ll(start); break;  // Create a new linked list
            case 9: return 0;  // Exit the program
            default: printf("Invalid choice, try again.\n");
        }
    }

    return 0;
}
```

Output:


C:\Users\temp\Desktop\ll1.exe
1. Insert at beg
2. Insert at last
3. Insert at pos
4. Delete beg
5. Delete last
6. Delete pos
7. Display
8. Create Linked List
9. Exit
Enter your choice: 8
Enter values for the nodes. Enter -1 to stop.
Enter value: 1
Enter value: 2
Enter value: 3
Enter value: -1
Enter your choice: 1
Enter the value to be inserted at the beginning:
100
Enter your choice: 2
Enter the value to be inserted at last:
200
Enter your choice: 7
List: 100 1 2 3 200
Enter your choice: 3
Enter value to be inserted:
300
Enter position to be inserted:
3
Enter your choice: 7
List: 100 1 300 2 3 200
Enter your choice: 4
Enter your choice: 7
List: 1 300 2 3 200
Enter your choice: 5
Enter your choice: 7
List: 1 300 2 3
Enter your choice: 6
Enter the position to be deleted: 2
Enter your choice: 7
List: 1 2 3
Enter your choice: 9
Process returned 0 (0x0) execution time : 81.960 s
Press any key to continue.

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.

Program:

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

// Node structure
struct Node {
    int val;
    struct Node* next;
};

// Create a new node
struct Node* newNode(int val) {
    struct Node* n = (struct Node*)malloc(sizeof(struct Node));
    n->val = val;
    n->next = NULL;
    return n;
}

// Insert a node at the end
struct Node* insert(struct Node* head, int val) {
    struct Node* n = newNode(val);
    if (!head) return n;
    struct Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = n;
```

```
    return head;
}


struct Node* sort(struct Node* head) { if (!head || !head->next) return head;
struct Node *i, *j; int temp;
for (i = head; i; i = i->next) {
for (j = i->next; j; j = j->next) {
if (i->val > j->val) {
temp = i->val; i->val = j->val; j->val = temp;
}
}
 } return head;
 }

// Reverse the list
struct Node* reverse(struct Node* head) {
    struct Node *prev = NULL, *current = head, *next;
    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

// Concatenate two lists
struct Node* concatenate(struct Node* list1, struct Node* list2) {
```

```
    if (!list1) return list2;
    struct Node* temp = list1;
    while (temp->next) temp = temp->next;
    temp->next = list2;
    return list1;
}

// Print the list
void print(struct Node* head) {
    while (head) {
        printf("%d -> ", head->val);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice, val;



        printf("\n1. Insert to List 1\n2. Insert to List 2\n3. Sort List 1\n4. Sort List 2\n");
        printf("5. Reverse List 1\n6. Reverse List 2\n7. Concatenate Lists\n8. Print List
1\n9. Print List 2\n10. Exit\n");


         while (1) {printf("Enter choice: ");
         scanf("%d", &choice);


        switch (choice) {
```

```
case 1:
    printf("Enter value to insert in List 1: ");
    scanf("%d", &val);
    list1 = insert(list1, val);
    break;
case 2:
    printf("Enter value to insert in List 2: ");
    scanf("%d", &val);
    list2 = insert(list2, val);
    break;
case 3:
    list1 = sort(list1);
    printf("List 1 sorted.\n");
    break;
case 4:
    list2 = sort(list2);
    printf("List 2 sorted.\n");
    break;
case 5:
    list1 = reverse(list1);
    printf("List 1 reversed.\n");
    break;
case 6:
    list2 = reverse(list2);
    printf("List 2 reversed.\n");
    break;
case 7:
    list1 = concatenate(list1, list2);
    printf("Lists concatenated.\n");
```

```
            break;
        case 8:
            printf("List 1: ");
            print(list1);
            break;
        case 9:
            printf("List 2: ");
            print(list2);
            break;
        case 10:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}
```

Output:

C:\Users\temp\Desktop\Sort,reverse,concat.exe

```
1. Insert to List 1
2. Insert to List 2
3. Sort List 1
4. Sort List 2
5. Reverse List 1
6. Reverse List 2
7. Concatenate Lists
8. Print List 1
9. Print List 2
10. Exit
Enter choice: 1
Enter value to insert in List 1: 1
Enter choice: 1
Enter value to insert in List 1: 2
Enter choice: 1
Enter value to insert in List 1: 3
Enter choice: 8
List 1: 1 -> 2 -> 3 -> NULL
Enter choice: 2
Enter value to insert in List 2: 7
Enter choice: 2
Enter value to insert in List 2: 8
Enter choice: 2
Enter value to insert in List 2: 9
Enter choice: 9
List 2: 7 -> 8 -> 9 -> NULL
Enter choice: 5
List 1 reversed.
Enter choice: 8
List 1: 3 -> 2 -> 1 -> NULL
Enter choice: 6
List 2 reversed.
Enter choice: 9
List 2: 9 -> 8 -> 7 -> NULL
Enter choice: 3
List 1 sorted.
Enter choice: 8
List 1: 1 -> 2 -> 3 -> NULL
Enter choice: 4
List 2 sorted.
Enter choice: 9
List 2: 7 -> 8 -> 9 -> NULL
Enter choice: 7
Lists concatenated.
Enter choice: 8
List 1: 1 -> 2 -> 3 -> 7 -> 8 -> 9 -> NULL
Enter choice: 10
Exiting...

Process returned 0 (0x0)   execution time : 78.070 s
Press any key to continue.
```

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

Program:

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

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

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Stack functions
struct Node* push(struct Node* top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = top;
    return newNode;
}

struct Node* pop(struct Node* top, int* poppedData) {
    if (top == NULL) {
        *poppedData = -1;
```

```
        return NULL;
    }
    struct Node* temp = top;
    *poppedData = temp->data;
    top = temp->next;
    free(temp);
    return top;
}

// Queue functions
struct Node* enqueue(struct Node* rear, int data) {
    struct Node* newNode = createNode(data);
    if (rear) rear->next = newNode;
    return newNode;
}

struct Node* dequeue(struct Node* front, int* dequeuedData) {
    if (front == NULL) {
        *dequeuedData = -1;
        return NULL;
    }
    struct Node* temp = front;
    *dequeuedData = temp->data;
    front = temp->next;
    free(temp);
    return front;
}

// Display functions
```

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

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



int main() {
    struct Node* stackTop = NULL;
```

```
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;
    int choice, value;

    printf("1: Push  2: Pop  3: Enqueue  4: Dequeue  5: Show Stack  6: Show Queue
7: Exit\n");

    while (1) {
        printf("\nChoice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Value: ");
                scanf("%d", &value);
                stackTop = push(stackTop, value);
                break;

            case 2: {
                int poppedData;
                stackTop = pop(stackTop, &poppedData);
                if (poppedData == -1) printf("Stack: Empty\n");
                else printf("Popped: %d\n", poppedData);
                break;
            }

            case 3:
                printf("Value: ");
                scanf("%d", &value);
                queueRear = enqueue(queueRear, value);
```

```
            if (queueFront == NULL) queueFront = queueRear; // Initialize front if
queue was empty
            break;

        case 4: {
            int dequeuedData;
            queueFront = dequeue(queueFront, &dequeuedData);
            if (queueFront == NULL) queueRear = NULL; // Reset rear if queue is now
empty
            if (dequeuedData == -1) printf("Queue: Empty\n");
            else printf("Dequeued: %d\n", dequeuedData);
            break;
        }

        case 5:
            displayStack(stackTop);
            break;

        case 6:
            displayQueue(queueFront);
            break;

        case 7:
            printf("Exit\n");
            exit(0);

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

```
    return 0;
}
```

Output:

```
1: Push  2: Pop  3: Enqueue  4: Dequeue  5: Show Stack  6: Show Queue  7: Exit

Choice: 1
Value: 120

Choice: 1
Value: 130

Choice: 1
Value: 140

Choice: 5
Stack: 140 130 120

Choice: 2
Popped: 140

Choice: 2
Popped: 130

Choice: 5
Stack: 120

Choice: 2
Popped: 120

Choice: 5
Stack: Empty

Choice: 3
Value: 100

Choice: 3
Value: 200

Choice: 3
Value: 400

Choice: 6
Queue: 100 200 400

Choice: 4
Dequeued: 100
```

```
Choice: 5
Stack: Empty

Choice: 3
Value: 100

Choice: 3
Value: 200

Choice: 3
Value: 400

Choice: 6
Queue: 100 200 400

Choice: 4
Dequeued: 100

Choice: 6
Queue: 200 400

Choice: 4
Dequeued: 200

Choice: 4
Dequeued: 400

Choice: 6
Queue: Empty

Choice: 7
Exit

Process returned 0 (0x0)   execution time : 103.393 s
Press any key to continue.
```

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

Program:

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

struct node *head = NULL;
struct node *insert_left(struct node *head,int data, int val);
struct node *delete(struct node *head,int val);
struct node *create_ll(struct node *head);
struct node *createnode(int data);

void display_ll(struct node *head);

struct node *createnode(int data1){
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = data1;
    newnode -> next = NULL;
    newnode -> prev = NULL;
    return newnode;
```

```
}
struct node *create_ll(struct node *head){
    int e_data;
    head = NULL;
    struct node* ptr = NULL;
    while(1){
            printf("Enter data: ");
            scanf("%d",&e_data);
            if(e_data == -1){
                break;
            }
            struct node *a_node = createnode(e_data);
            if(head == NULL){
                head = a_node;
                ptr = head;
            }
            else{
                ptr->next = a_node;
                a_node -> prev = ptr;
                ptr = ptr->next;

            }
        }
        return head;
}

void display(struct node *head){
    struct node *ptr = head;
    printf("Linked list is : ");
```

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

struct node *insert_left(struct node *head, int data, int val){
    struct node *ptr = head;
    if(head == NULL){
        printf("No element in the list \n");
        return head;
    }
    while(ptr!=NULL && ptr->data != val){
        ptr = ptr ->next;
    }
    if(ptr==NULL){
        printf("%d not found",val );
        return head;

    }
    else{
        struct node *a_node = createnode(data);
        a_node ->next = ptr;
        if(ptr->prev != NULL){
            a_node->prev = ptr->prev;
            ptr->prev->next = a_node;

        }
```

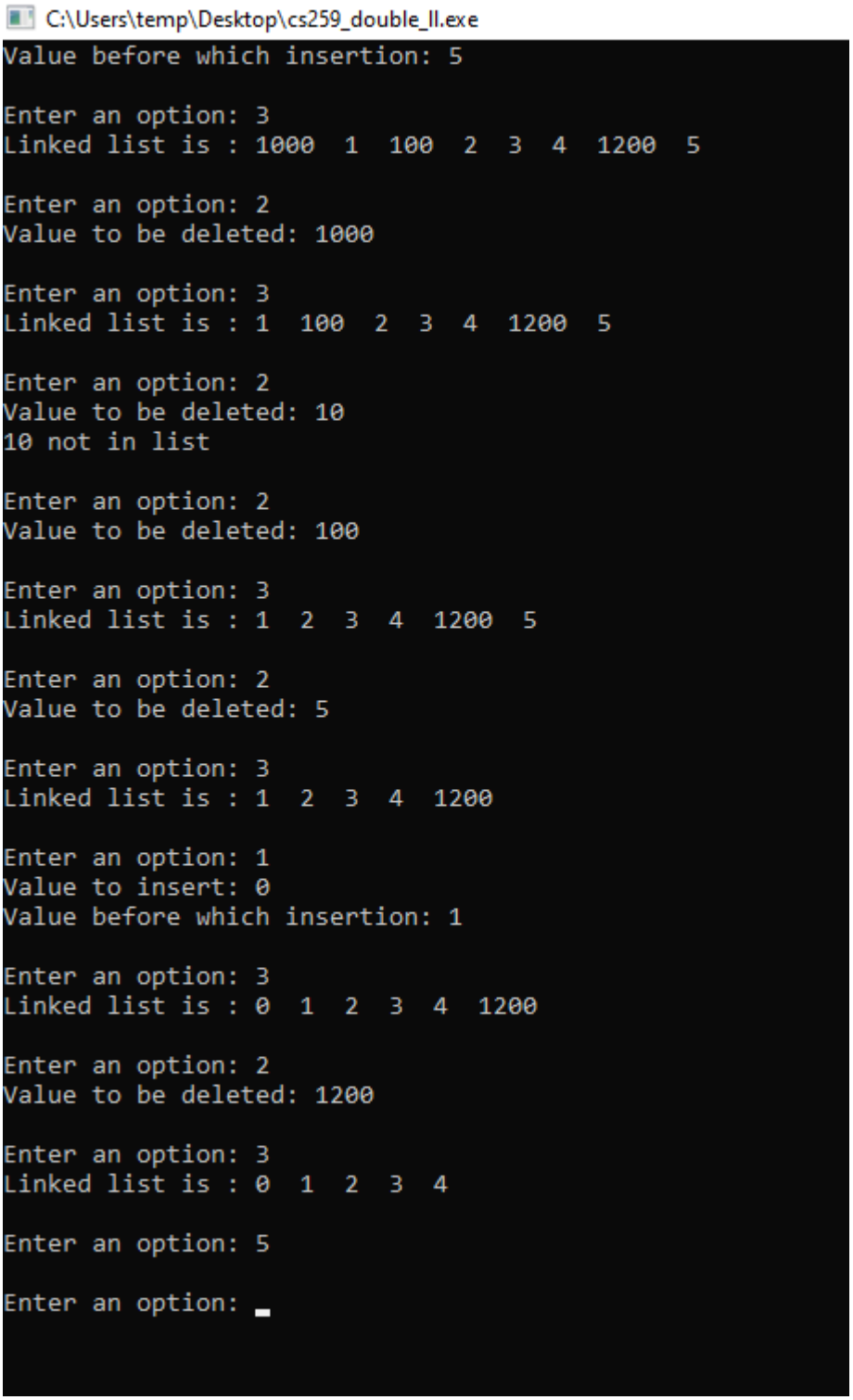
```
        else{
            head = a_node;
        }
        ptr->prev = a_node;
        return head;
    }

}
struct node *delete(struct node* head, int data1){
    struct node* ptr = head;
    if(head == NULL){
            printf("Empty\n");
                return head;
       }
    else{
        while(ptr!=NULL && ptr->data!=data1){
            ptr = ptr->next;
        }
        if(ptr==NULL){
            printf("%d not in list\n",data1);
        }
        else{
            if(ptr->prev!=NULL){
                ptr->prev->next = ptr->next;
            }
            else{
                head = ptr->next;
            }
            if(ptr->next != NULL){
```

```
            ptr->next->prev = ptr->prev;
        }
    }
    free(ptr);
    return head;

  }
}

void main(){
  printf("1. Insert left\n 2. Delete\n3.display\n4.create list\n 5. exit");
  int opt;
  printf("\nEnter an option: ");
  scanf("%d",&opt);
  while(1){
    switch(opt){
      int val,data ;
      case 1:
          printf("Value to insert: "); scanf("%d",&data);
          printf("Value before which insertion: "); scanf("%d",&val);
          head = insert_left(head,data,val);
          break;
      case 2:
          printf("Value to be deleted: "); scanf("%d",&val);
          head = delete(head,val);
          break;
      case 3: display(head);break;
      case 4: head = create_ll(head); break;
      case 5: break;
```

```
        }
         printf("\nEnter an option: ");
    scanf("%d",&opt);
    }
}
```

Output:

```
1. Insert left
 2. Delete
3.display
4.create list
 5. exit
Enter an option: 4
Enter data: 1
Enter data: 2
Enter data: 3
Enter data: 4
Enter data: 5
Enter data: -1

Enter an option: 3
Linked list is : 1  2  3  4  5

Enter an option: 1
Value to insert: 100
Value before which insertion: 2

Enter an option: 3
Linked list is : 1  100  2  3  4  5

Enter an option: 1
Value to insert: 1000
Value before which insertion: 1

Enter an option: 3
Linked list is : 1000  1  100  2  3  4  5

Enter an option: 1
Value to insert: 1200
Value before which insertion: 5

Enter an option: 3
Linked list is : 1000  1  100  2  3  4  1200  5

Enter an option: 2
Value to be deleted: 1000

Enter an option: 3
Linked list is : 1  100  2  3  4  1200  5

Enter an option: 2
Value to be deleted: 10
10 not in list

Enter an option: 2
Value to be deleted: 100

Enter an option: 3
Linked list is : 1  2  3  4  1200  5

Enter an option: 2
Value to be deleted: 5

Enter an option: 3
Linked list is : 1  2  3  4  1200

Enter an option: 1
Value to insert: 0
Value before which insertion: 1
```


C:\Users\temp\Desktop\cs259_double_ll.exe
Value before which insertion: 5
Enter an option: 3
Linked list is : 1000 1 100 2 3 4 1200 5
Enter an option: 2
Value to be deleted: 1000
Enter an option: 3
Linked list is : 1 100 2 3 4 1200 5
Enter an option: 2
Value to be deleted: 10
10 not in list
Enter an option: 2
Value to be deleted: 100
Enter an option: 3
Linked list is : 1 2 3 4 1200 5
Enter an option: 2
Value to be deleted: 5
Enter an option: 3
Linked list is : 1 2 3 4 1200
Enter an option: 1
Value to insert: 0
Value before which insertion: 1
Enter an option: 3
Linked list is : 0 1 2 3 4 1200
Enter an option: 2
Value to be deleted: 1200
Enter an option: 3
Linked list is : 0 1 2 3 4
Enter an option: 5
Enter an option:

LAB PROGRAM 8

Write a program

a) ToconstructabinarySearchtree.

b) To traverse the tree using all the methods i.e., inorder, preorder and post order

c) To display the elements in the tree.

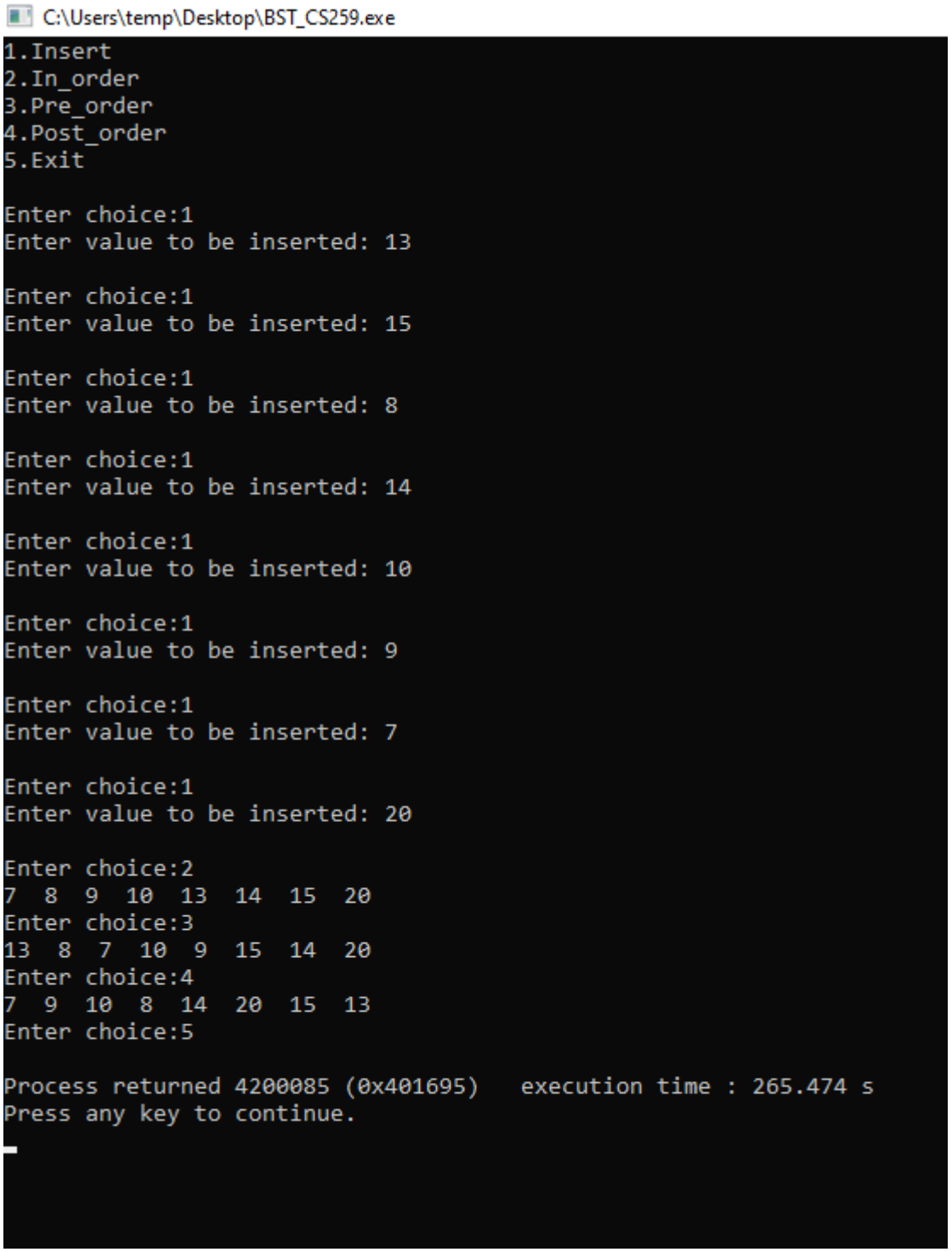
Program:

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

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

struct node* root = NULL;
struct node* createnode(int data){
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode -> data = data;
    newnode -> left = NULL;
    newnode -> right = NULL;
    return newnode;
}

struct node* insert_node(struct node* root, int val){
    if (root == NULL) {
        return createnode(val);
    }
```

```
    if (val < root->data) {
        root->left = insert_node(root->left, val);
    } else {
        root->right = insert_node(root->right, val);
    }
    return root;
}

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

        printf("%d  ",root->data);
        pre_order(root->left);
        pre_order(root->right);
    }
    return root;
}
struct node* post_order(struct node* root){
    if(root!=NULL){
        post_order(root->left);
        post_order(root->right);
```

```
        printf("%d  ",root->data);
    }
    return root;
}

void main(){
    printf("1.Insert\n2.In_order\n3.Pre_order\n4.Post_order\n5.Exit\n");
    int flag = 0;
    while(flag == 0){
        int opt;
        printf("\nEnter choice:");
        scanf("%d",&opt);
        switch(opt){
            case 1: printf("Enter value to be inserted: ");
                    int val;
                    scanf("%d",&val);
                    root = insert_node(root,val);
                    break;
            case 2: root = in_order(root); break;
            case 3: root = pre_order(root); break;
            case 4: root = post_order(root); break;
            case 5: flag = 1;break;
        }
    }
}
```

Output:


C:\Users\temp\Desktop\BST_CS259.exe
1.Insert
2.In_order
3.Pre_order
4.Post_order
5.Exit
Enter choice:1
Enter value to be inserted: 13
Enter choice:1
Enter value to be inserted: 15
Enter choice:1
Enter value to be inserted: 8
Enter choice:1
Enter value to be inserted: 14
Enter choice:1
Enter value to be inserted: 10
Enter choice:1
Enter value to be inserted: 9
Enter choice:1
Enter value to be inserted: 7
Enter choice:1
Enter value to be inserted: 20
Enter choice:2
7 8 9 10 13 14 15 20
Enter choice:3
13 8 7 10 9 15 14 20
Enter choice:4
7 9 10 8 14 20 15 13
Enter choice:5
Process returned 4200085 (0x401695) execution time : 265.474 s
Press any key to continue.

LAB PROGRAM 9

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

Program:

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

#define MAX 20

struct Queue {
    int items[MAX];
    int front, rear;
};

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

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

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX - 1) {
        printf("Queue Overflow\n");
        return;
```

```
    }
    if (q->front == -1) q->front = 0;
    q->items[++q->rear] = value;
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue Underflow\n");
        return -1;
    }
    int item = q->items[q->front];
    if (q->front == q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return item;
}
void BFS(int adj_graph[MAX][MAX], int n, int start) {
    bool visited[MAX] = {false};
    struct Queue* q = createQueue();

    printf("BFS Traversal: ");
    visited[start] = true;
    enqueue(q, start);

    while (!isEmpty(q)) {
        int current = dequeue(q);
        printf("%d ", current);
```

```
        for (int i = 0; i < n; i++) {
            if (adj_graph[current][i] == 1 && !visited[i]) {
                visited[i] = true;
                enqueue(q, i);
            }
        }
    }
    printf("\n");
}
int main() {
    int n, start, adj_graph[MAX][MAX];

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

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

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

    BFS(adj_graph, n, start);
    return 0;
}
```

Output:

C:\Users\temp\Desktop\BFS_cs259.exe

```
Enter the number of vertices: 5
Enter the adjacency matrix:
0 0 1 1 0
0 0 1 0 0
1 1 0 0 1
1 0 0 0 1
0 0 1 1 0
Enter the starting vertex: 1
BFS Traversal: 1 2 0 4 3

Process returned 0 (0x0)   execution time : 175.076 s
Press any key to continue.
```

C:\Users\temp\Desktop\BFS_cs259.exe

```
Enter the number of vertices: 5
Enter the adjacency matrix:
0 0 1 1 0
0 0 1 0 0
1 1 0 0 1
1 0 0 0 1
0 0 1 1 0
Enter the starting vertex: 0
BFS Traversal: 0 2 3 1 4

Process returned 0 (0x0)   execution time : 24.341 s
Press any key to continue.
```

C:\Users\temp\Desktop\BFS_cs259.exe

```
Enter the number of vertices: 5
Enter the adjacency matrix:
0 0 1 1 0
0 0 1 0 0
1 1 0 0 1
1 0 0 0 1
0 0 1 1 0
Enter the starting vertex: 4
BFS Traversal: 4 2 3 0 1

Process returned 0 (0x0)   execution time : 20.373 s
Press any key to continue.
```

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

Program:

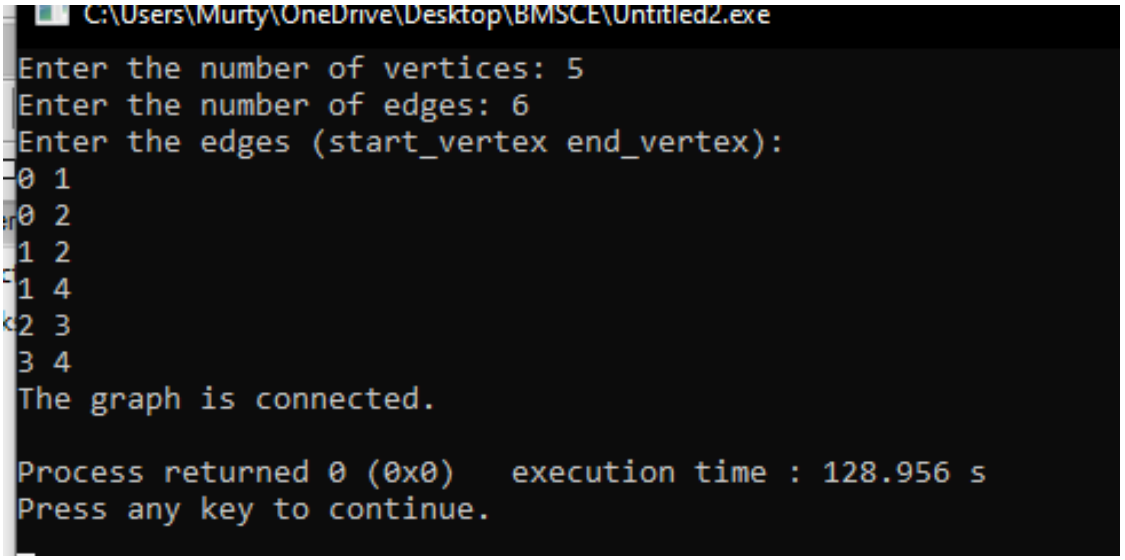
```
#include <stdio.h>
#include <stdbool.h>

#define MAX 100

int adjMatrix[MAX][MAX];
bool visited[MAX];
int stack[MAX];
int top = -1;

void push(int vertex) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = vertex;
}

int pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[top--];
}

void dfsUsingStack(int startVertex, int numVertices) {
```

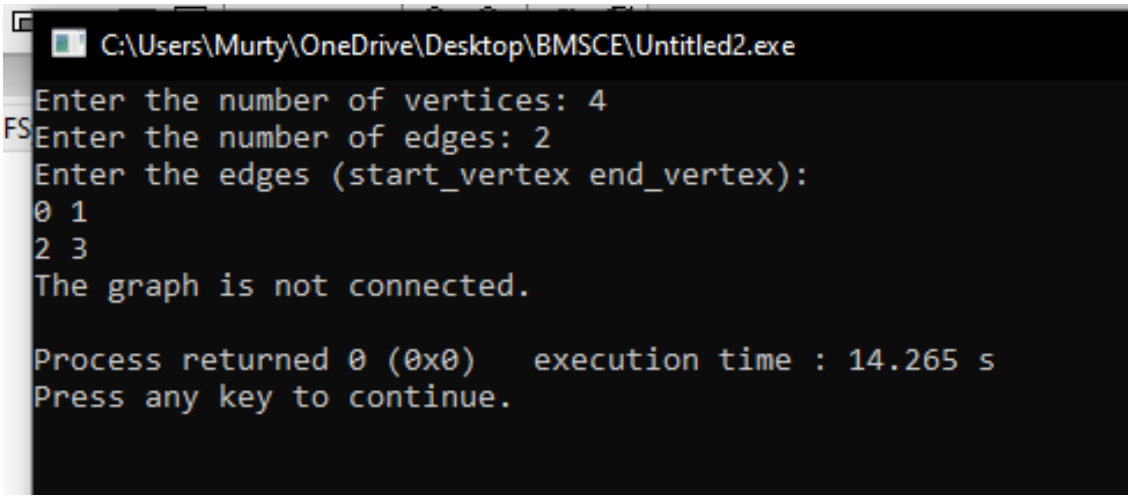
```
    push(startVertex);
    visited[startVertex] = true;

    while (top != -1) {
        int currentVertex = pop();

        for (int i = 0; i < numVertices; i++) {
            if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
                push(i);
                visited[i] = true;
            }
        }
    }
}

bool isConnected(int numVertices) {
    for (int i = 0; i < numVertices; i++) {
        visited[i] = false;
    }

    dfsUsingStack(0, numVertices);

    for (int i = 0; i < numVertices; i++) {
        if (!visited[i]) {
            return false;
        }
    }
    return true;
}
```

```
int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

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

    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            adjMatrix[i][j] = 0;
        }
    }

    printf("Enter the edges (start_vertex end_vertex):\n");
    for (int i = 0; i < numEdges; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
    }

    if (isConnected(numVertices)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }
```

```
    return 0;
}
```

Output:





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.

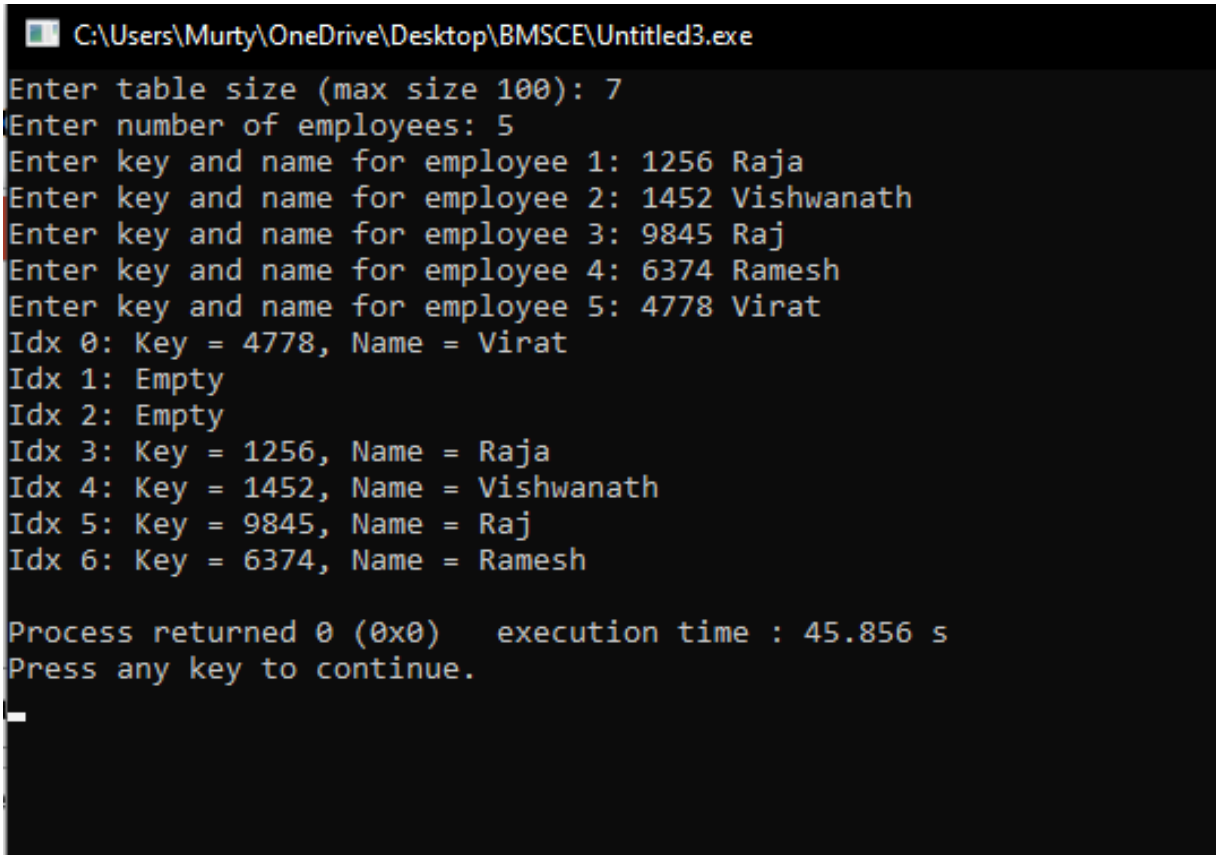
Program:

```
#include <stdio.h>

#define MAX 100


struct Employee {
    int k;
    char n[50];
};


struct Employee ht[MAX];
int ts;


void init() {
    for (int i = 0; i < MAX; i++) ht[i].k = -1;
}


int hash(int k) {
    return k % ts;
}
```

```
void insert(int k, char n[]) {
    int idx = hash(k);
    while (ht[idx].k != -1) {
        idx = (idx + 1) % ts;
    }
    ht[idx].k = k;
    for (int i = 0; n[i] != '\0' && i < 49; i++) {
        ht[idx].n[i] = n[i];
    }
    ht[idx].n[49] = '\0';
}

void display() {
    for (int i = 0; i < ts; i++) {
        if (ht[i].k != -1)
            printf("Idx %d: Key = %d, Name = %s\n", i, ht[i].k, ht[i].n);
        else
            printf("Idx %d: Empty\n", i);
    }
}

int main() {
    int n;
    printf("Enter table size (max size %d): ", MAX);
    scanf("%d", &ts);

    if (ts > MAX) ts = MAX;

    init();
```

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

    for (int i = 0; i < n; i++) {
        int k;
        char name[50];
        printf("Enter key and name for employee %d: ", i + 1);
        scanf("%d", &k);
        getchar();
        gets(name);
        insert(k, name);
    }

    display();

    return 0;
}
```

Output:

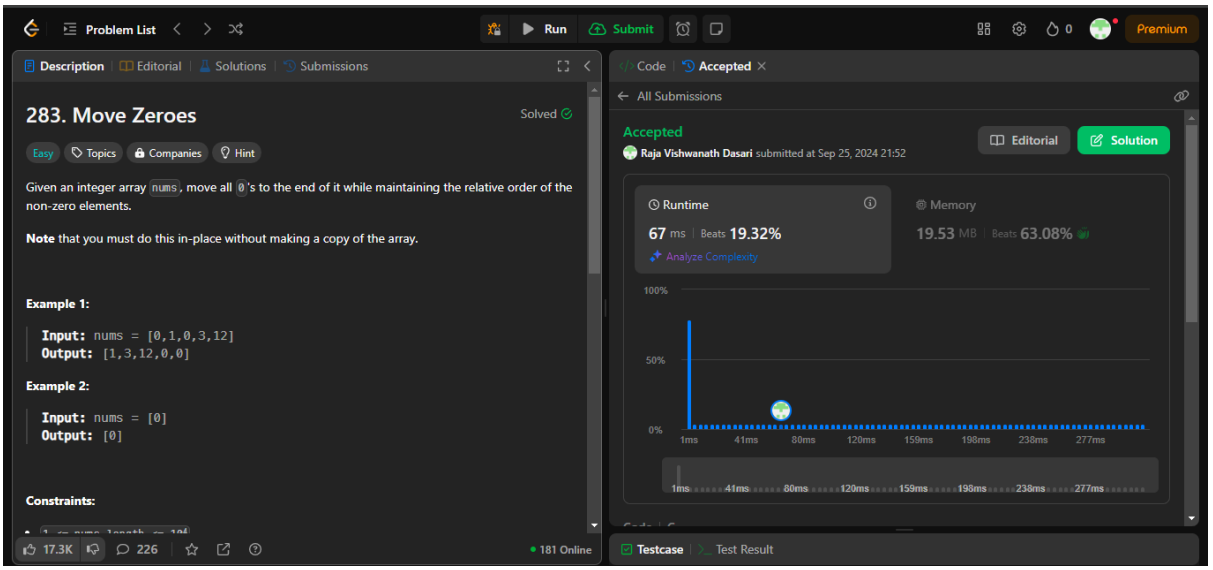

C:\Users\Murty\OneDrive\Desktop\BMSCE\Untitled3.exe
Enter table size (max size 100): 7
Enter number of employees: 5
Enter key and name for employee 1: 1256 Raja
Enter key and name for employee 2: 1452 Vishwanath
Enter key and name for employee 3: 9845 Raj
Enter key and name for employee 4: 6374 Ramesh
Enter key and name for employee 5: 4778 Virat
Idx 0: Key = 4778, Name = Virat
Idx 1: Empty
Idx 2: Empty
Idx 3: Key = 1256, Name = Raja
Idx 4: Key = 1452, Name = Vishwanath
Idx 5: Key = 9845, Name = Raj
Idx 6: Key = 6374, Name = Ramesh
Process returned 0 (0x0) execution time : 45.856 s
Press any key to continue.

LEETCODE - MOVE ZEROES

Program:

```
void moveZeroes(int* nums, int numsSize) {
    int ind = 0;

    for (int i = 0; i < numsSize; i++) {
        if (nums[i] != 0) {
            nums[ind] = nums[i];
            ind++;
        }
    }

    for (int i = ind; i < numsSize; i++) {
        nums[i] = 0;
    }
}
```

Output:

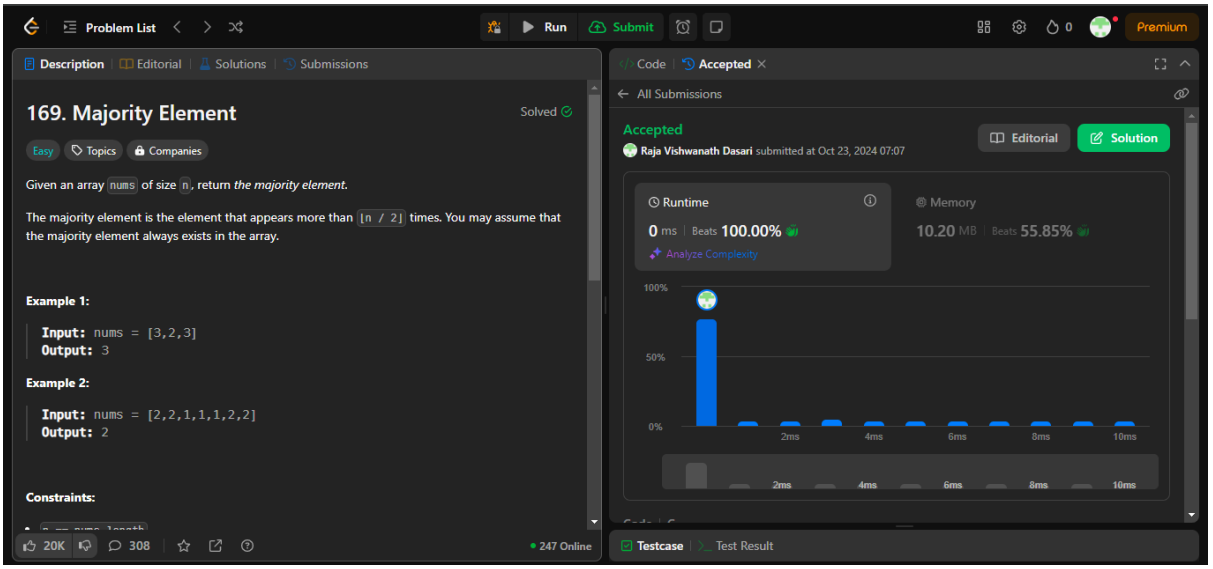


LEETCODE - MAJORITY ELEMENT

Program:

```
int majorityElement(int* nums, int numsSize) {
    int candidate = 0;
    int count = 0;

    for (int i = 0; i < numsSize; i++) {
        if (count == 0) {
            candidate = nums[i];
        }
        if (nums[i] == candidate) {
            count++;
        } else {
            count--;
        }
    }
   return candidate;
}
```

Output:



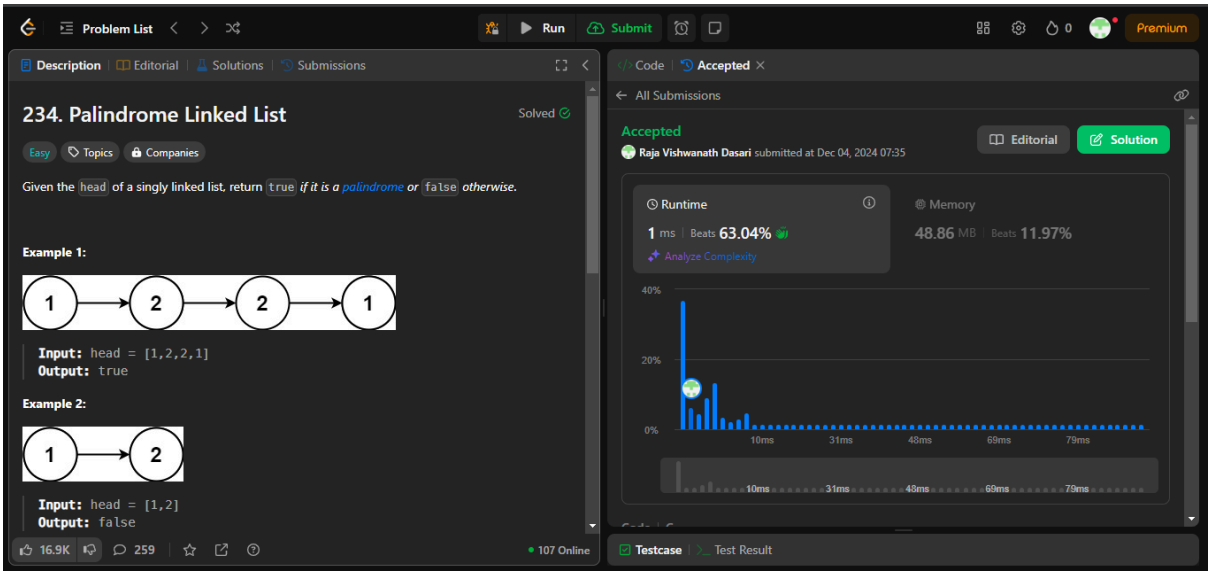
LEETCODE - PALINDROME LINKED LIST

Program:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
bool isPalindrome(struct ListNode* head) {

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

    int size = 0;
    struct ListNode* temp = head;
    while (temp) {
        size++;
        temp = temp->next;
    }

    int* values = (int*)malloc(size * sizeof(int));
    temp = head;
    for (int i = 0; i < size; i++) {
        values[i] = temp->val;
        temp = temp->next;
    }
```

```
    int start = 0, end = size - 1;
    while (start < end) {
        if (values[start] != values[end]) {
            free(values);
            return false;
        }
        start++;
        end--;
    }

    free(values); // Clean up memory.
    return true;
}
```

Output:

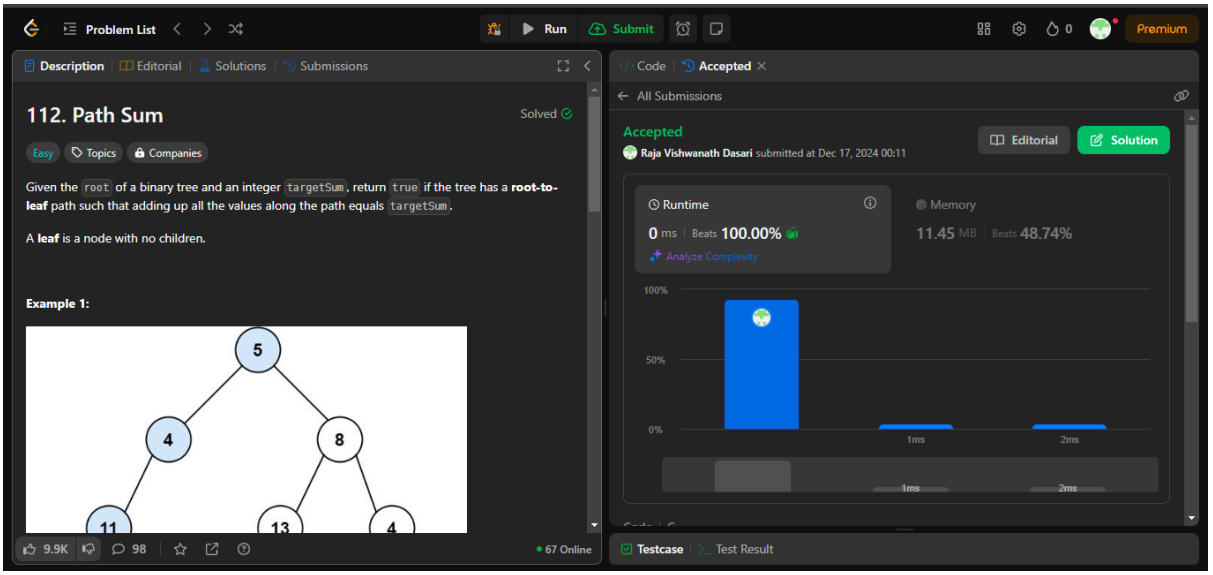


LEETCODE - PATH SUM

Program:

```
#include <stdbool.h>

// Definition for a binary tree node.
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

bool checkPathSum(struct TreeNode* root, int currentSum, int targetSum) {

    if (root == NULL) {
        return false;
    }
    currentSum += root->val;
    if (root->left == NULL && root->right == NULL) {
        return currentSum == targetSum;
    }

    return checkPathSum(root->left, currentSum, targetSum) ||
           checkPathSum(root->right, currentSum, targetSum);
}

bool hasPathSum(struct TreeNode* root, int targetSum) {
    return checkPathSum(root, 0, targetSum);
}
```

Output:


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
112. Path Sum
Solved
Easy
Topics
Companies
Given the root of a binary tree and an integer targetSum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals targetSum.
A leaf is a node with no children.
Example 1:
5
4
8
11
13
4
9.9K
98
67 Online
Code
Accepted
All Submissions
Accepted
Raja Vishwanath Dasari submitted at Dec 17, 2024 00:11
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
11.45 MB | Beats 48.74%
100%
50%
0%
1ms
2ms
Testcase
Test Result

HACKERRANK- GAME OF TWO STACKS

Program:

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'twoStacks' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER maxSum
 *  2. INTEGER_ARRAY a
 *  3. INTEGER_ARRAY b
 */
```

```
int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {
   int max_i = 0, max_j = 0;
    int runsum_a = 0, runsum_b = 0;

    while (max_i < a_count && runsum_a + a[max_i] <= maxSum) {
        runsum_a += a[max_i];
        max_i++;
    }

    while (max_j < b_count && runsum_b + b[max_j] <= maxSum) {
        runsum_b += b[max_j];
        max_j++;
    }

    int prefix_sum_a[max_i + 1], prefix_sum_b[max_j + 1];
    prefix_sum_a[0] = 0;
    prefix_sum_b[0] = 0;

    for (int x = 1; x <= max_i; x++) {
        prefix_sum_a[x] = prefix_sum_a[x - 1] + a[x - 1];
    }

    for (int y = 1; y <= max_j; y++) {
        prefix_sum_b[y] = prefix_sum_b[y - 1] + b[y - 1];
    }

    int max_count = 0;
```

```
    for (int i1 = 0; i1 <= max_i; i1++) {
        int sum_a = prefix_sum_a[i1];

        int remaining_sum = maxSum - sum_a;

        int left = 0, right = max_j, j1 = 0;
        while (left <= right) {
            int mid = (left + right) / 2;
            if (prefix_sum_b[mid] <= remaining_sum) {
                j1 = mid;
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        int total_elements = i1 + j1;
        if (total_elements > max_count) {
            max_count = total_elements;
        }
    }

    return max_count;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
```

```
int g = parse_int(ltrim(rtrim(readline())));

for (int g_itr = 0; g_itr < g; g_itr++) {
    char** first_multiple_input = split_string(rtrim(readline()));

    int n = parse_int(*(first_multiple_input + 0));

    int m = parse_int(*(first_multiple_input + 1));

    int maxSum = parse_int(*(first_multiple_input + 2));

    char** a_temp = split_string(rtrim(readline()));

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

    for (int i = 0; i < n; i++) {
        int a_item = parse_int(*(a_temp + i));

        *(a + i) = a_item;
    }

    char** b_temp = split_string(rtrim(readline()));

    int* b = malloc(m * sizeof(int));

    for (int i = 0; i < m; i++) {
        int b_item = parse_int(*(b_temp + i));

        *(b + i) = b_item;
```

```
        }

        int result = twoStacks(maxSum, n, a, m, b);

        fprintf(fptr, "%d\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
```

```
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
```

```
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
```

```
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
```

```
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Output: