**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**SAANVI S (IBM23CS282)**

**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 SAANVI S (**1BM23CS282)**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.

**Prof. Lakshmi Neelima M** **Dr. Kavitha Sooda**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 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. | 4 |
| 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).  Demonstration of account creation on LeetCode platform Program - Leetcode platform. | 6 |
| 3 | a)WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions  b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messagesfor queue empty and queue overflow conditions | 8 |
| 4 | a)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) Deletion of first element, specified element and last element in the list.  Display the contents of the linked list.  b)LEETCODE : MAJORITY ELEMENT | 15 |
| 5 | LEETCODE : (game of stacks – hackarank) | 22 |
| 6 | a)WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists  b)WAP to Implement Single Link List to simulate Stack & Queue Operations. | 22 |
| 7 | a)WAP to Implement doubly link list with primitive operations   * Create a doubly linked list. * Insert a new node to the left of the node. * Delete the node based on a specific value * Display the contents of the list   b) LEETCODE :(List Palindrome) | 32 |
| 8 | a.Write a program To construct a binary Search tree. To traverse the tree using all the methods i.e., in-order, preorder and post order To display the elements in the tree.  2) LEET CODE: (hasPathSum) | 37 |
| 9 | a) Write a program to traverse a graph using BFS method  b) Write a program to check whether given graph is connected or not using DFS method. | 39 |
| 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. | 42 |

**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 SIZE 5

int STACK[SIZE];

int top = -1;

void push(int value) {

if (top == SIZE - 1){

printf("Stack overflow!\n");

}

else {

top++;

STACK[top]=value;

printf("%d pushed onto stack\n",value);

}

}

void pop() {

if(top==-1) {

printf("Stack underflow!!\n");

}

else{

int value = STACK[top];

top--;

printf("%d popped successfully!\n",value);

}

}

void display(){

if(top==-1){

printf("stack is empty\n");

}

else{

printf("stack elements :\n");

for(int i=top;i>=0;i--){

printf("%d",STACK[i]);

printf("\n");

}

}

}

int main() {

int choice,a;

while(1) {

printf("enter 1-push, 2-pop, 3-display, 4-exit :");

scanf("%d",&choice);

switch(choice) {

case 1:

printf("Enter value :");

scanf("%d",&a);

push(a);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default :

printf("Invalid choice! enter again");

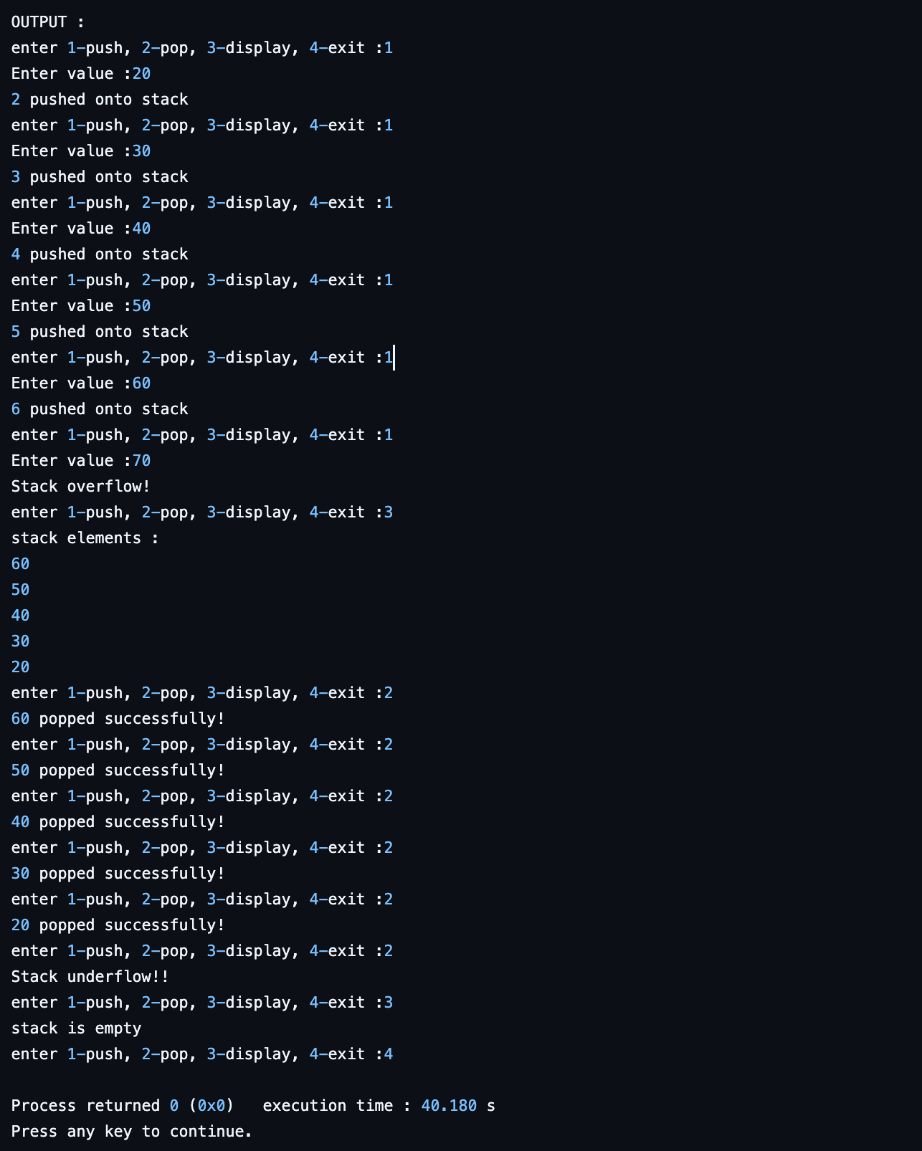
}

}

return 0;

}

**Output:**



**Lab program 2:**

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

#include <stdio.h>

#include <string.h>

int index1 = 0, pos = 0, top = -1, length;

char symbol, temp, infix[20], postfix[20], stack[20];

void infixToPostfix();

void push(char symbol);

char pop();

int pred(char symbol);

int main() {

printf("Enter infix expression:\n");

scanf("%s", infix);

infixToPostfix();

printf("\nInfix expression: %s", infix);

printf("\nPostfix expression: %s\n", postfix);

return 0;

}

void infixToPostfix() {

length = strlen(infix);

push('#'); // Push an initial dummy character to the stack

while (index1 < length) {

symbol = infix[index1];

switch (symbol) {

case '(':

push(symbol);

break;

case ')':

temp = pop();

while (temp != '(') {

postfix[pos++] = temp;

temp = pop();

}

break;

case '+':

case '-':

case '\*':

case '/':

case '^':

while (pred(stack[top]) >= pred(symbol)) {

temp = pop();

postfix[pos++] = temp;

}

push(symbol);

break;

default:

postfix[pos++] = symbol;

}

index1++;

}

while (top > 0) {

temp = pop();

postfix[pos++] = temp;

}

postfix[pos] = '\0';

}

void push(char symbol) {

top = top + 1;

stack[top] = symbol;

}

char pop() {

char symb;

symb = stack[top];

top = top - 1;

return symb;

}

int pred(char symbol) {

int p;

switch (symbol) {

case '^':

p = 3;

break;

case '\*':

case '/':

p = 2;

break;

case '+':

case '-':

p = 1;

break;

case '(':

p = 0;

break;

case '#':

p = -1;

break;

default:

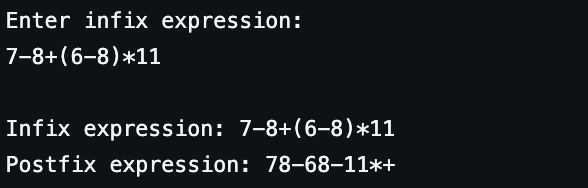
p = -1;

}

return p;

}

**Output:**



**b) LEETCODE : movezeroes**

void moveZeroes(int\* nums, int numsSize)

{

int lindex=0;

for(int index=0;index < numsSize;index++)

{

if(nums[index]!=0)

{

nums[lindex]=nums[index];

lindex++;

}

}

for(int i = lindex;i < numsSize;i++)

{

nums[i]=0;

}

}

**Lab program 3:**

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

#include <stdio.h>

#define MAX 5

int queue[MAX];

int front = -1;

int rear = -1;

void enqueue(int value) {

if (rear == MAX - 1) {

printf("Queue is full! Overflow !\n");

} else {

if (front == -1) {

front = 0;

}

rear++;

queue[rear] = value;

printf("Inserted %d\n", value);

}

}

int dequeue() {

if (front == -1 || front > rear) {

printf("Queue is empty! Underflow!\n");

return -1;

} else {

int item = queue[front];

front++;

if (front > rear) {

front = rear = -1;

}

printf("Deleted %d\n", item);

return item;

}

}

void display() {

if (front == -1 || front > rear) {

printf("Queue is empty!\n");

} else {

printf("Queue elements are: ");

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

printf("\n");

}

}

int main() {

int choice, value;

while (1) {

printf("\nQueue Operations:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

return 0;

default:

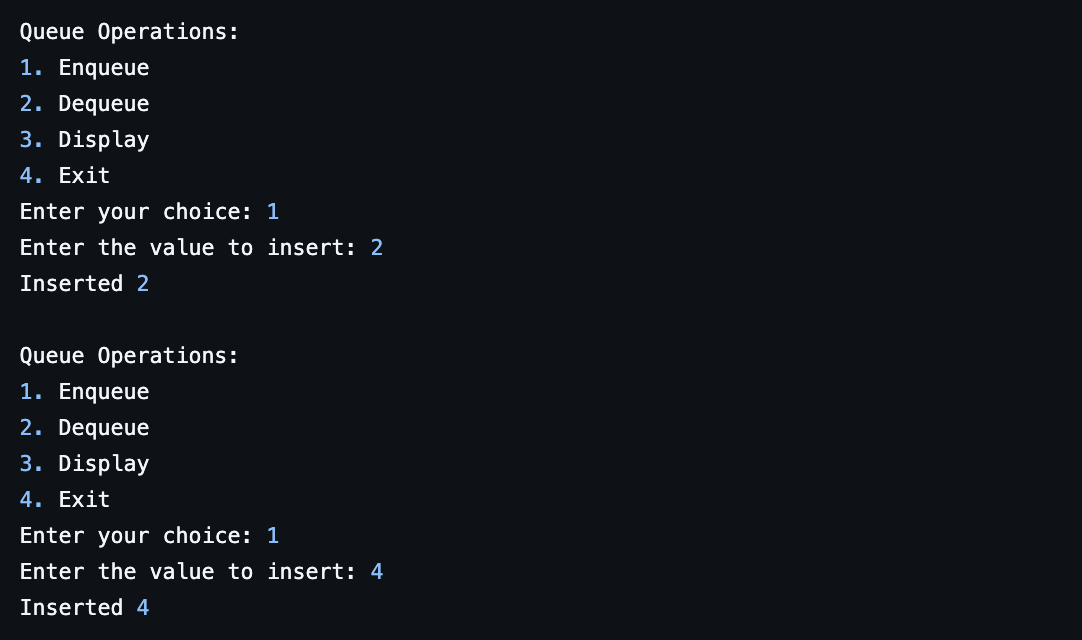
printf("Invalid choice! Please try again.\n");

}

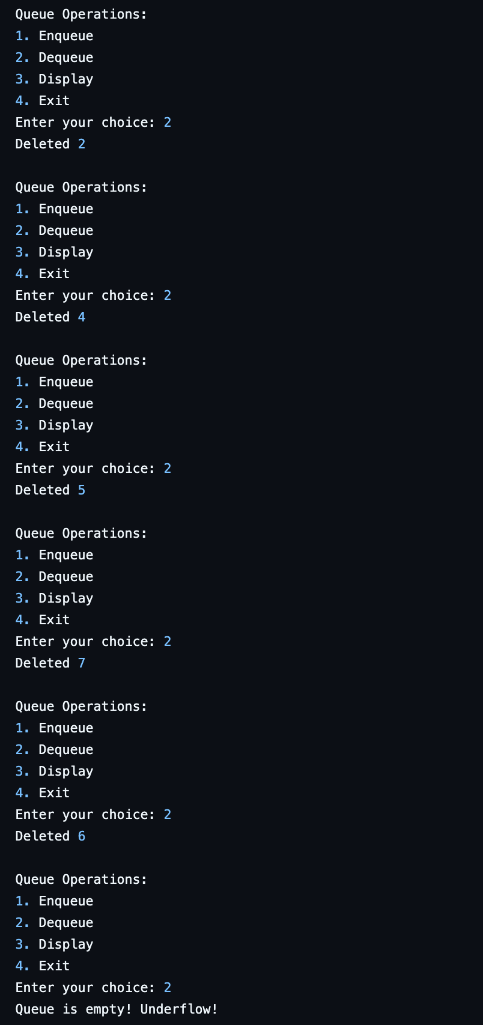
}

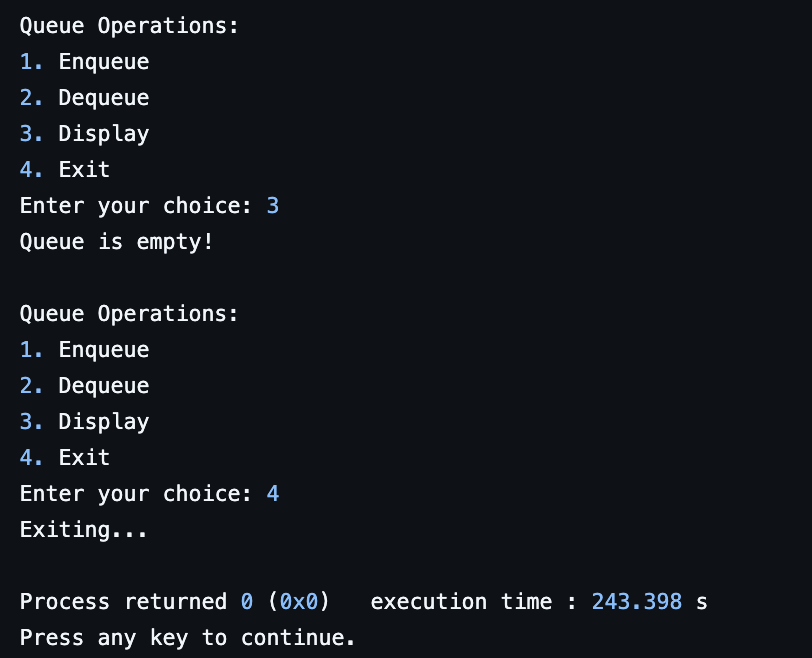
}

**Output:**









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

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

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

int isFull() {

if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))

return 1;

return 0;

}

int isEmpty() {

if (front == -1)

return 1;

return 0;

}

}

void enqueue(int element) {

if (isFull()) {

printf("\nQueue is full!!\n");

} else {

if (front == -1)

front = 0;

rear = (rear + 1) % SIZE;

items[rear] = element;

printf("\n%d is inserted into the queue.\n", element);

}

}

int dequeue() {

int element;

if (isEmpty()) {

printf("\nQueue is empty!!\n");

return -1;

} else {

element = items[front];

if (front == rear) {

front = -1;

rear = -1;

} else {

front = (front + 1) % SIZE;

}

printf("\n%d is deleted from the queue.\n", element);

return element;

}

}

void display() {

int i;

if (isEmpty()) {

printf("\nQueue is empty!!\n");

} else {

printf("\nFront position: %d\n", front);

printf("Queue elements: ");

for (i = front; i != rear; i = (i + 1) % SIZE) {

printf("%d ", items[i]);

}

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

}

}

int main() {

int choice, element;

while (1) {

printf("\n\*\*\*\*\*\* Circular Queue Operations \*\*\*\*\*\*\n");

printf("1. Enqueue\n"); printf("2. Dequeue\n");

printf("3. Display\n"); printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the element to insert: ");

scanf("%d", &element);

enqueue(element);

break;

case 2:

element = dequeue();

if (element != -1)

printf("%d element is deleted.\n", element);

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("\nInvalid choice! Please try again.\n");

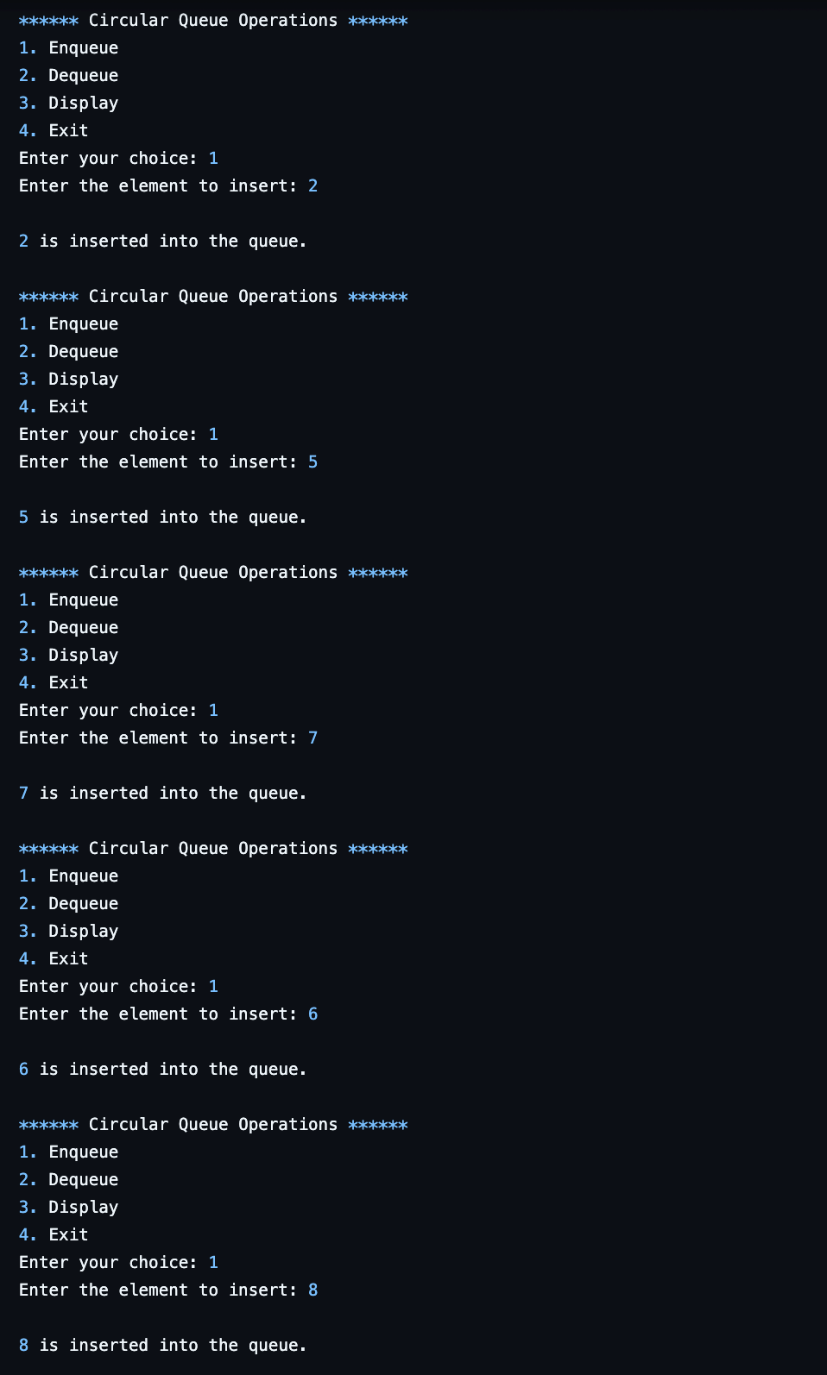
}

}

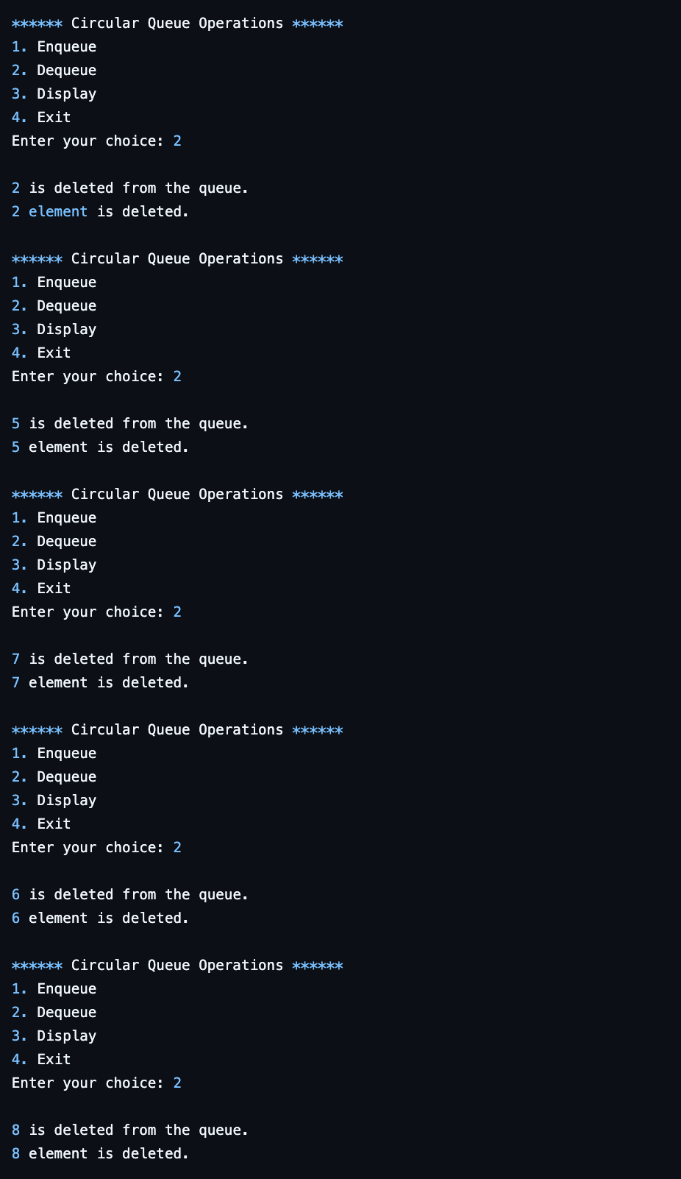
return 0;

}

**Output:**









**Lab program 4:**

**a)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) Deletion of first element, specified element and last element in the list.**

**Display the contents of the linked list.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

struct node \*start = NULL;

struct node \*create\_ll(struct node \*start) {

struct node \*new\_node, \*ptr;

int num;

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

printf("Enter the data: ");

scanf("%d", &num);

while (num != -1) {

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

new\_node->next = NULL;

if (start == NULL) {

start = new\_node;

} else {

ptr = start;

while (ptr->next != NULL) {

ptr = ptr->next;

}

ptr->next = new\_node;

}

printf("Enter the data: ");

scanf("%d", &num);

}

return start;

}

struct node \*display(struct node \*start) {

struct node \*ptr;

ptr = start;

while (ptr != NULL) {

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

ptr = ptr->next;

}

printf("NULL\n");

return start;

}

struct node \*insert\_beg(struct node \*start) {

struct node \*new\_node;

int num;

printf("Enter the data: ");

scanf("%d", &num);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

new\_node->next = start;

start = new\_node;

return start;

}

struct node \*insert\_end(struct node \*start) {

struct node \*ptr, \*new\_node;

int num;

printf("Enter the data: ");

scanf("%d", &num);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

new\_node->next = NULL;

ptr = start;

while (ptr->next != NULL) {

ptr = ptr->next;

}

ptr->next = new\_node;

return start;

}

struct node \*insert\_before(struct node \*start) {

struct node \*new\_node, \*ptr, \*preptr;

int num, val;

printf("Enter the data: ");

scanf("%d", &num);

printf("Enter the value before which the data has to be inserted: ");

scanf("%d", &val);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

ptr = start;

while (ptr->data != val) {

preptr = ptr;

ptr = ptr->next;

}

preptr->next = new\_node;

new\_node->next = ptr;

return start;

}

struct node \*insert\_after(struct node \*start) {

struct node \*new\_node, \*ptr;

int num, val;

printf("Enter the data: ");

scanf("%d", &num);

printf("Enter the value after which the data has to be inserted: ");

scanf("%d", &val);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

ptr = start;

while (ptr->data != val) {

ptr = ptr->next;

}

new\_node->next = ptr->next;

ptr->next = new\_node;

return start;

}

struct node \*delete\_beg(struct node \*start) {

struct node \*ptr;

ptr = start;

start = start->next;

free(ptr);

return start;

}

struct node \*delete\_end(struct node \*start) {

struct node \*ptr, \*preptr;

ptr = start;

while (ptr->next != NULL) {

preptr = ptr;

ptr = ptr->next;

}

preptr->next = NULL;

free(ptr);

return start;

}

struct node \*delete\_node(struct node \*start) {

struct node \*ptr, \*preptr;

int val;

printf("Enter the value of the node to be deleted: ");

scanf("%d", &val);

ptr = start;

if (ptr->data == val) {

start = delete\_beg(start);

return start;

} else {

while (ptr->data != val) {

preptr = ptr;

ptr = ptr->next;

}

preptr->next = ptr->next;

free(ptr);

return start;

}

}

struct node \*delete\_after(struct node \*start) {

struct node \*ptr, \*preptr;

int val;

printf("Enter the value after which the node has to be deleted: ");

scanf("%d", &val);

ptr = start;

while (preptr->data != val) {

preptr = ptr;

ptr = ptr->next;

}

preptr->next = ptr->next;

free(ptr);

return start;

}

int main() {

int option;

do {

printf("\n\n \*\*\*\*\*MAIN MENU\*\*\*\*\*");

printf("\n 1: Create a list");

printf("\n 2: Display the list");

printf("\n 3: Add a node at the beginning");

printf("\n 4: Add a node at the end");

printf("\n 5: Add a node before a given node");

printf("\n 6: Add a node after a given node");

printf("\n 7: Delete a node from the beginning");

printf("\n 8: Delete a node from the end");

printf("\n 9: Delete a given node");

printf("\n 10: Delete a node after a given node");

printf("\n 11: EXIT");

printf("\n\n Enter your option: ");

scanf("%d", &option);

switch (option) {

case 1: start = create\_ll(start);

printf("LINKED LIST CREATED");

break;

case 2: start = display(start);

break;

case 3: start = insert\_beg(start);

break;

case 4: start = insert\_end(start);

break;

case 5: start = insert\_before(start);

break;

case 6: start = insert\_after(start);

break;

case 7: start = delete\_beg(start);

break;

case 8: start = delete\_end(start);

break;

case 9: start = delete\_node(start);

break;

case 10: start = delete\_after(start);

break;

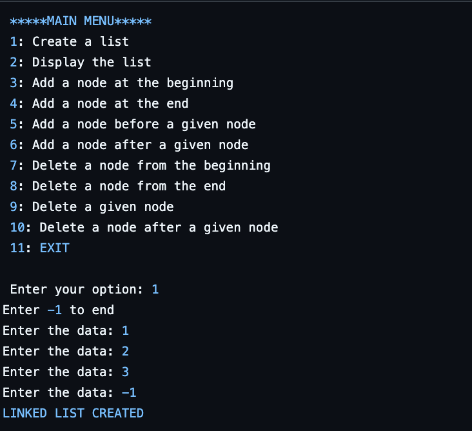
}

} while (option != 11);

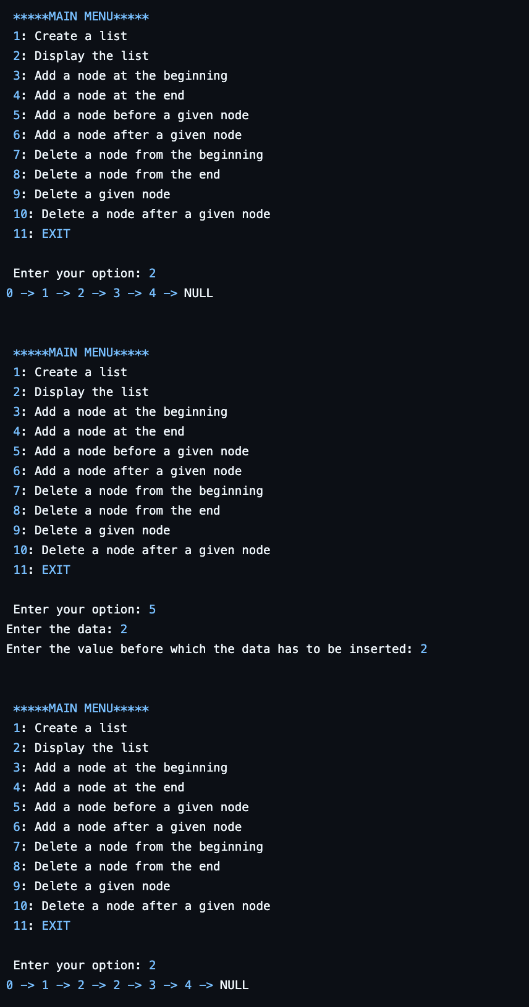
return 0;

}

**Output:**

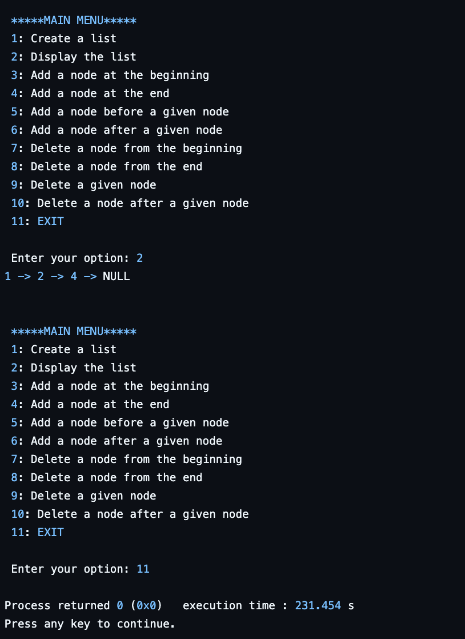












**b)LEETCODE : MAJORITY ELEMENT**

int majorityElement(int\* nums, int numsSize) {

int n = numsSize;

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

int count=0;

for(int j = 0; j<n;j++){

if(nums[j]==nums[i]){

count++;

}

}

if(count > n/2)

{

return nums[i];

}

}

return -1;

}

**Lab program 5: - LEETCODE : (game of stacks – hackarank)**

int twoStacks(int maxSum, int a\_count, int\* a, int b\_count, int\* b) {

int sumA = 0, sumB = 0;

int countA, countB;

for (countA = 0; countA < a\_count && sumA + a[countA] <= maxSum; countA++) {

sumA += a[countA];

}

int maxCount = countA;

for (countB = 0; countB < b\_count; countB++) {

sumB += b[countB];

for (; sumA + sumB > maxSum && countA > 0; countA--) {

sumA -= a[countA - 1];

}

if (sumA + sumB <= maxSum) {

maxCount = (countA + countB + 1 > maxCount) ? countA + countB + 1 : maxCount;

}

}

return maxCount;

}

**Lab program 6:**

**a)WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists**

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

#include <stdio.h>

#include <stdlib.h>

// Node structure for the linked list

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;

}

// Insert a node at the end of the linked list

void insertEnd(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

}

// Display the linked list

void displayList(struct Node\* head) {

while (head) {

printf("%d -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

// Sort the linked list

void sortList(struct Node\*\* head) {

if (\*head == NULL || (\*head)->next == NULL) return;

struct Node\* i;

struct Node\* j;

for (i = \*head; i->next; i = i->next) {

for (j = i->next; j; j = j->next) {

if (i->data > j->data) {

int temp = i->data;

i->data = j->data;

j->data = temp;

}

}

}

}

// Reverse the linked list

void reverseList(struct Node\*\* head) {

struct Node\* prev = NULL;

struct Node\* current = \*head;

struct Node\* next = NULL;

while (current) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head = prev;

}

// Concatenate two linked lists

void concatenateLists(struct Node\*\* head1, struct Node\*\* head2) {

if (\*head1 == NULL) {

\*head1 = \*head2;

return;

}

struct Node\* temp = \*head1;

while (temp->next) {

temp = temp->next;

}

temp->next = \*head2;

}

// Stack

void push(struct Node\*\* stack, int data) {

struct Node\* newNode = createNode(data);

newNode->next = \*stack;

\*stack = newNode;

}

int pop(struct Node\*\* stack) {

if (\*stack == NULL) {

printf("Stack underflow\n");

return -1;

}

int data = (\*stack)->data;

struct Node\* temp = \*stack;

\*stack = (\*stack)->next;

free(temp);

return data;

}

// Queue

void enqueue(struct Node\*\* queue, int data) {

insertEnd(queue, data);

}

int dequeue(struct Node\*\* queue) {

if (\*queue == NULL) {

printf("Queue underflow\n");

return -1;

}

int data = (\*queue)->data;

struct Node\* temp = \*queue;

\*queue = (\*queue)->next;

free(temp);

return data;

}

// display the stack

void displayStack(struct Node\* stack) {

if (stack == NULL) {

printf("Stack is empty\n");

return;

}

printf("Stack: ");

while (stack) {

printf("\n%d ", stack->data);

stack = stack->next;

}

printf("\n");

}

// display the queue

void displayQueue(struct Node\* queue) {

if (queue == NULL) {

printf("Queue is empty\n");

return;

}

printf("Queue: ");

while (queue) {

printf("%d ", queue->data);

queue = queue->next;

}

printf("\n");

}

// Main function

int main() {

struct Node\* list1 = NULL;

struct Node\* list2 = NULL;

struct Node\* stack = NULL;

struct Node\* queue = NULL;

int choice, value;

while (1) {

printf("\nMenu:\n");

printf("1) Insert into Linked List\n");

printf("2) Display Linked List\n");

printf("3) Sort Linked List\n");

printf("4) Reverse Linked List\n");

printf("5) Concatenate Two Linked Lists\n");

printf("6) Push to Stack\n");

printf("7) Pop from Stack\n");

printf("8) Enqueue to Queue\n");

printf("9) Dequeue from Queue\n");

printf("10) Display Stack\n");

printf("11) Display Queue\n");

printf("12) Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert into Linked List: ");

scanf("%d", &value);

insertEnd(&list1, value);

break;

case 2:

printf("Linked List: ");

displayList(list1);

break;

case 3:

sortList(&list1);

printf("Linked List sorted.\n");

break;

case 4:

reverseList(&list1);

printf("Linked List reversed.\n");

break;

case 5:

printf("Enter values for the second list (terminate with -1):\n");

while (1) {

scanf("%d", &value);

if (value == -1) break;

insertEnd(&list2, value);

}

concatenateLists(&list1, &list2);

printf("Lists concatenated.\n");

break;

case 6:

printf("Enter value to push onto Stack: ");

scanf("%d", &value);

push(&stack, value);

break;

case 7:

value = pop(&stack);

if (value != -1) {

printf("Popped from Stack: %d\n", value);

}

break;

case 8:

printf("Enter value to enqueue to Queue: ");

scanf("%d", &value);

enqueue(&queue, value);

break;

case 9:

value = dequeue(&queue);

if (value != -1) {

printf("Dequeued from Queue: %d\n", value);

}

break;

case 10:

displayStack(stack);

break;

case 11:

displayQueue(queue);

break;

case 12:

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid choice. Please try again.\n");

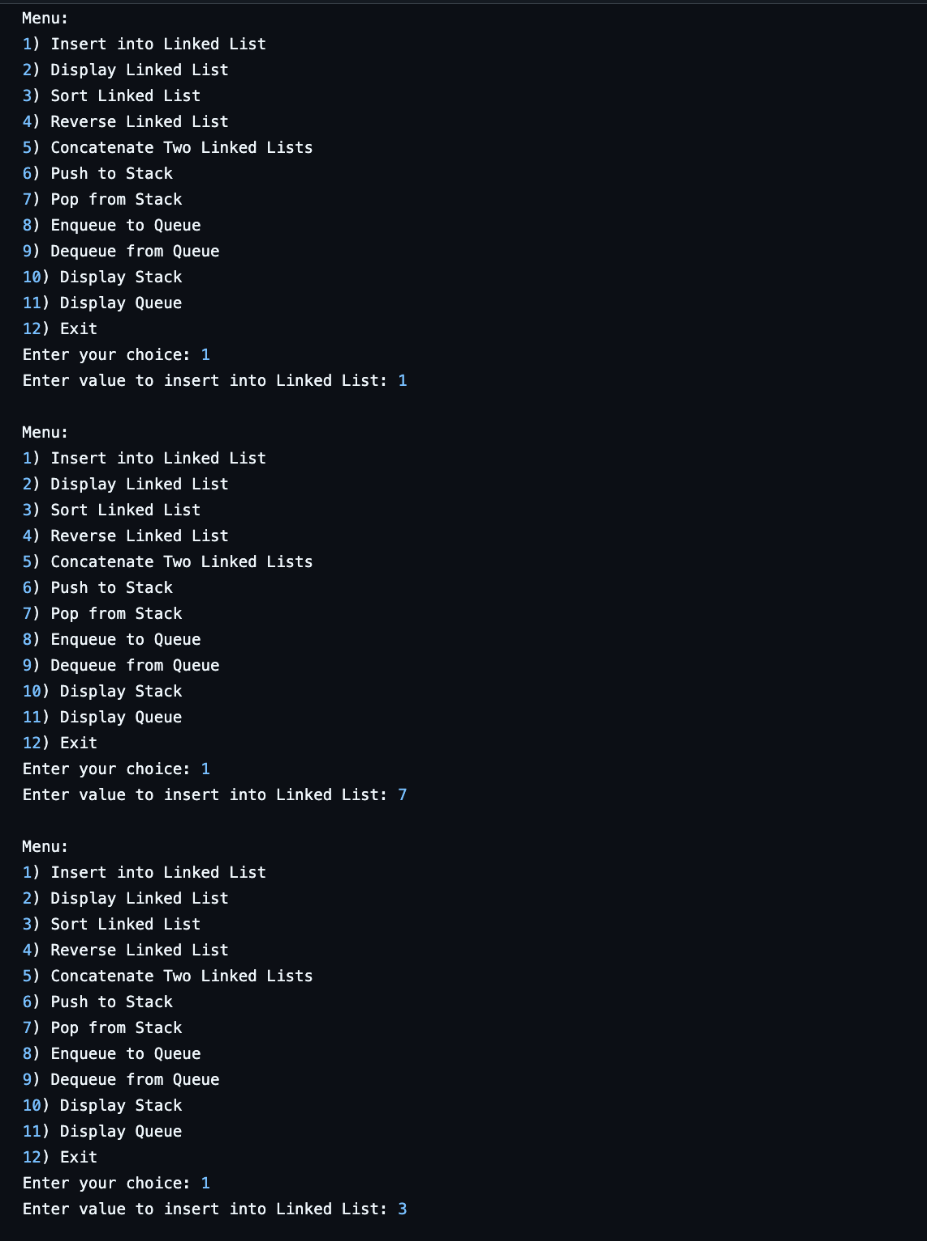
}

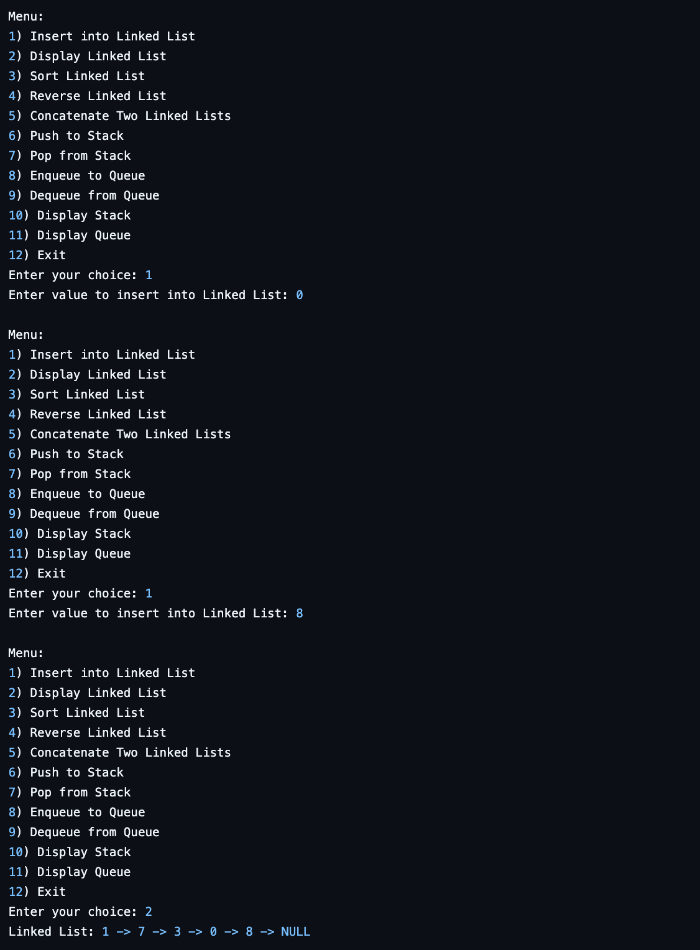
}

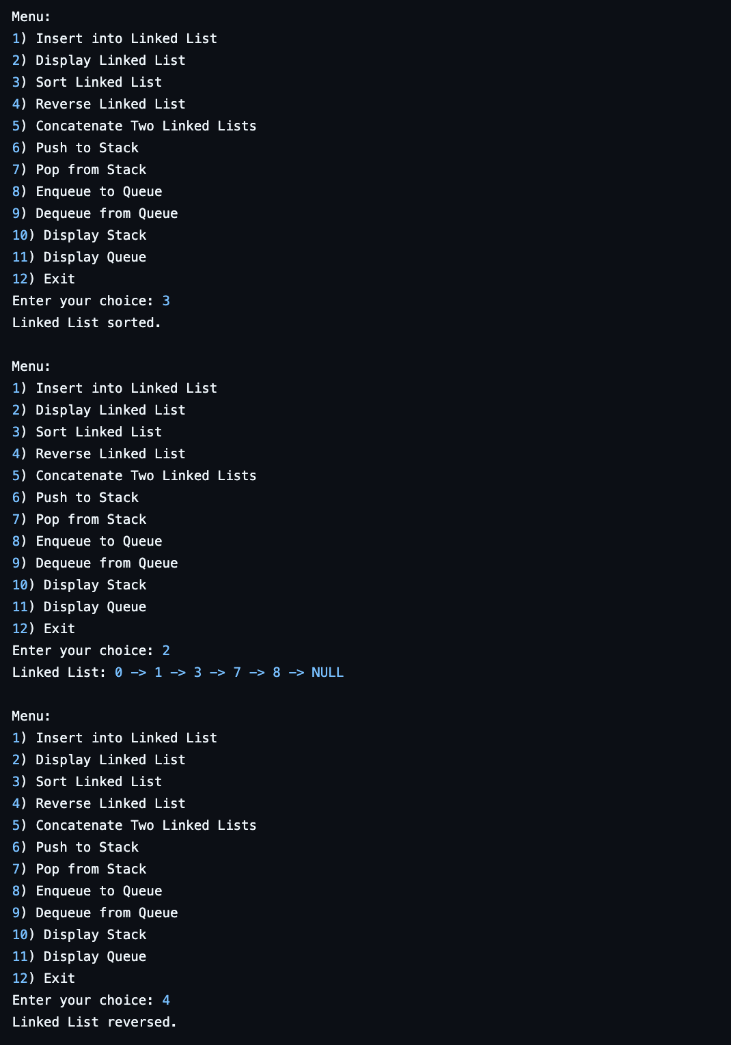
return 0;

}

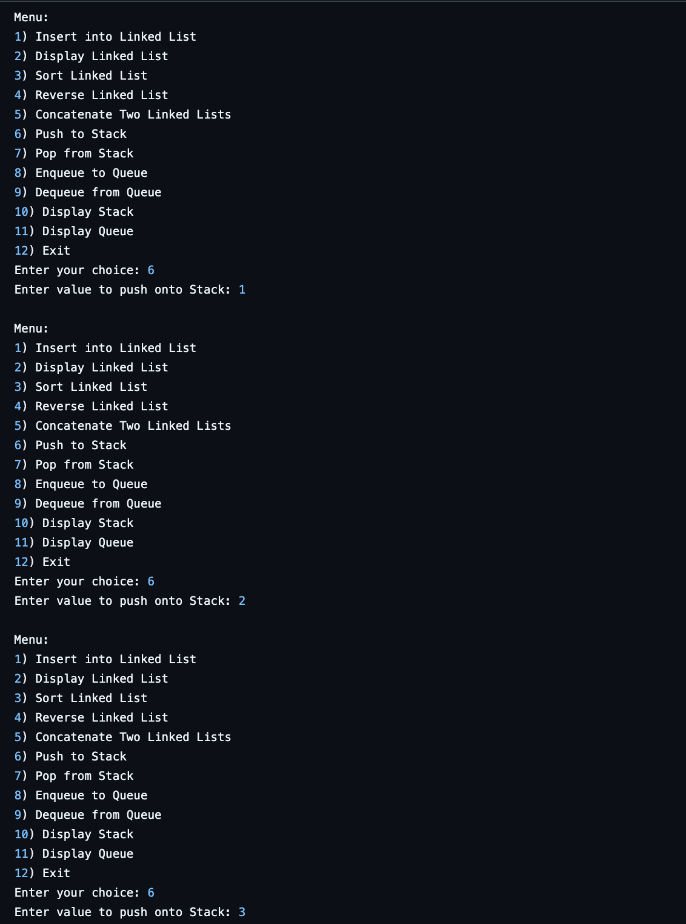
**Output:**

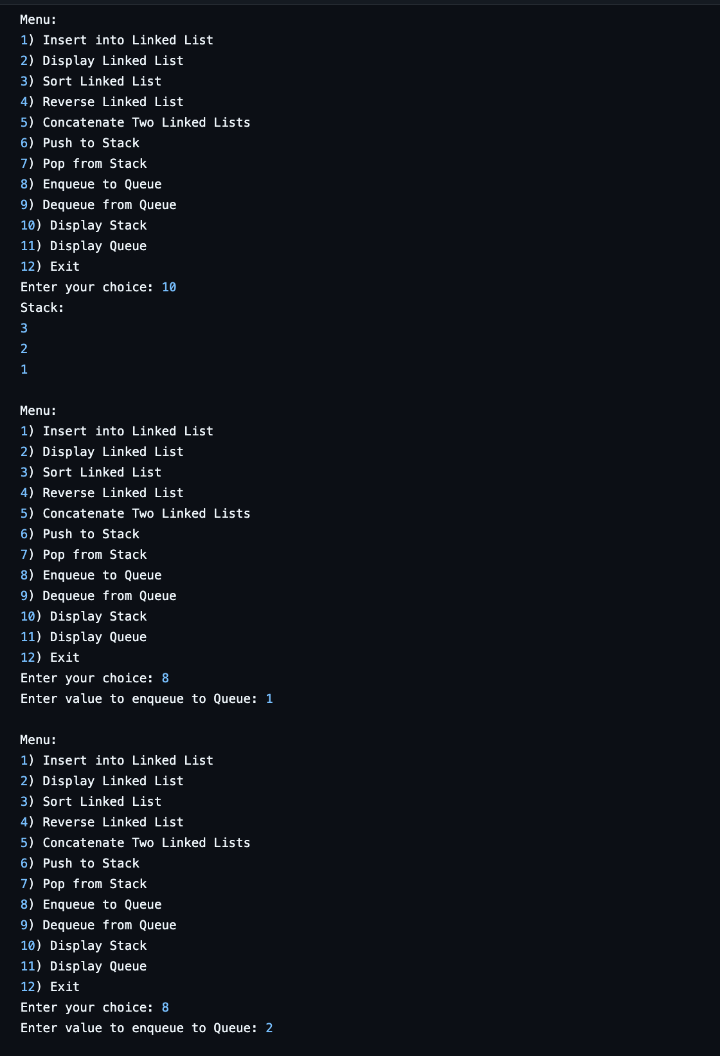


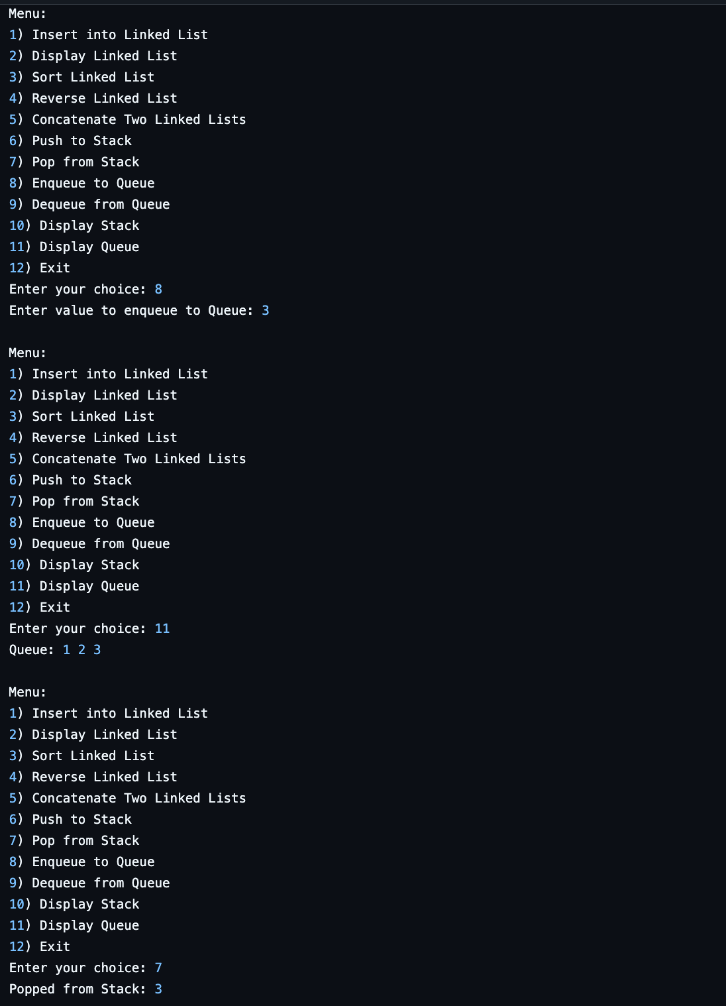


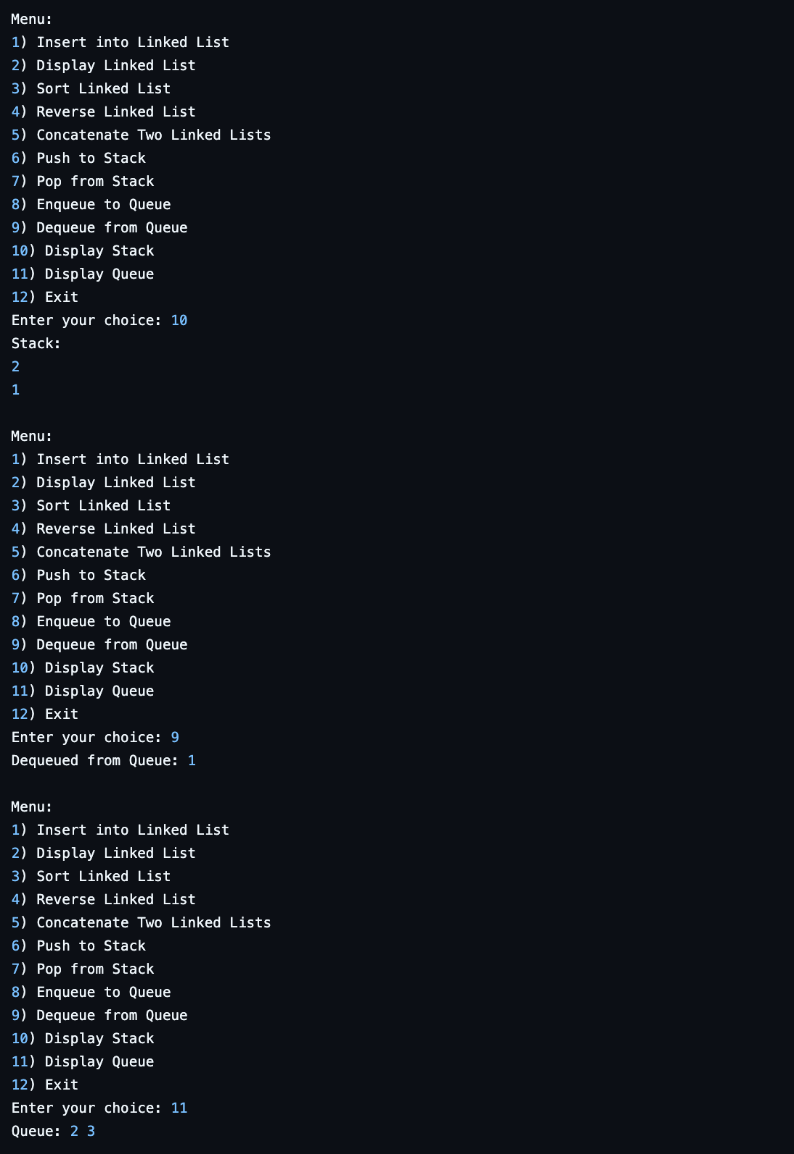














**Lab program 7:**

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

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

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

void append(struct Node\*\* head\_ref, int data){

struct Node\* newNode = createNode(data);

struct Node\* temp = \*head\_ref;

if (\*head\_ref == NULL){

\*head\_ref = newNode;

return;

}

while (temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

void insertLeft(struct Node\*\* head\_ref, int target, int data) {

struct Node\* temp = \*head\_ref;

while (temp != NULL && temp->data != target) {

temp = temp->next;

}

if (temp == NULL) {

printf("Node with value %d not found.\n", target);

return;

}

struct Node\* newNode = createNode(data);

newNode->next = temp;

newNode->prev = temp->prev;

if (temp->prev != NULL) {

temp->prev->next = newNode;

} else {

\*head\_ref = newNode;

}

temp->prev = newNode;

}

void deleteNode(struct Node\*\* head\_ref, int target) {

struct Node\* temp = \*head\_ref;

while (temp != NULL && temp->data != target) {

temp = temp->next;

}

if (temp == NULL) {

printf("Node with value %d not found.\n", target);

return;

}

if (temp->prev != NULL) {

temp->prev->next = temp->next;

} else {

\*head\_ref = temp->next;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

free(temp);

}

void display(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

printf("%d", temp->data);

if (temp->next != NULL) {

printf(" <-> ");

}

temp = temp->next;

}

printf("\n");

}

int main() {

struct Node\* head = NULL;

int choice, data, target;

do {

printf("\n-- Menu --\n");

printf("1. Append a node\n");

printf("2. Insert a node to the left of a given node\n");

printf("3. Delete a node by value\n");

printf("4. Display the list\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to append: ");

scanf("%d", &data);

append(&head, data);

break;

case 2:

printf("Enter the target value: ");

scanf("%d", &target);

printf("Enter the value to insert: ");

scanf("%d", &data);

insertLeft(&head, target, data);

break;

case 3:

printf("Enter the value to delete: ");

scanf("%d", &target);

deleteNode(&head, target);

break;

case 4:

printf("List contents: ");

display(head);

break;

case 5:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice. Please try again.\n");

}

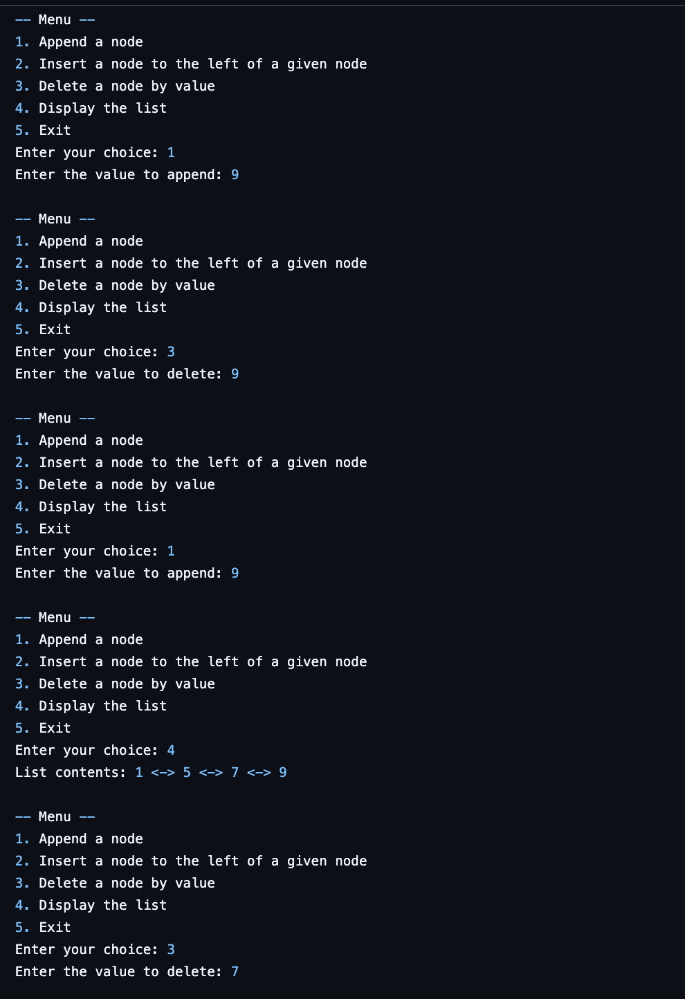
} while (choice != 5);

return 0;

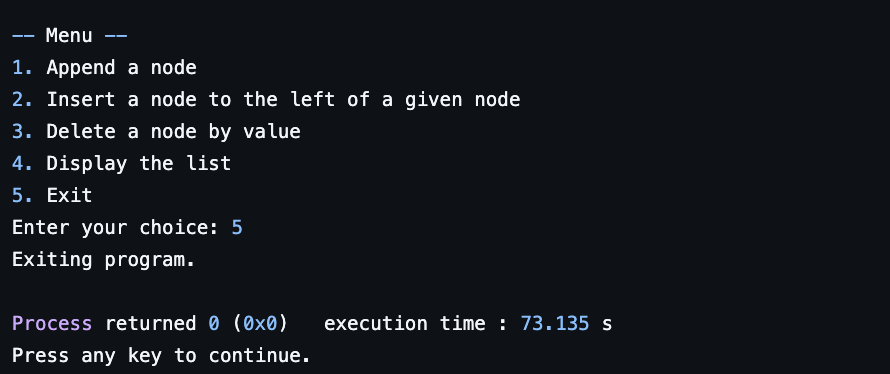
}

**Output:**









**b) LEETCODE :(List Palindrome)**

bool isPalindrome(struct ListNode\* head) {

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

return true;

}

int length = 0;

struct ListNode\* temp = head;

while (temp != NULL) {

length++;

temp = temp->next;

}

int values[length];

temp = head;

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

values[i] = temp->val;

temp = temp->next;

}

int left = 0;

int right = length - 1;

while (left < right) {

if (values[left] != values[right]) {

return false;

}

left++;

right--;

}

return true;

}

**Lab program 8:**

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

#include <stdio.h>

#include <malloc.h>

typedef struct BST {

int data;

struct BST \*left;

struct BST \*right;

} node;

node \*create() {

node \*temp;

printf("Enter data: ");

temp = (node \*)malloc(sizeof(node));

scanf("%d", &temp->data);

temp->left = temp->right = NULL;

return temp;

}

void insert(node \*root, node \*temp) {

if (temp->data < root->data) {

if (root->left != NULL)

insert(root->left, temp);

else

root->left = temp;

} else if (temp->data > root->data) {

if (root->right != NULL)

insert(root->right, temp);

else

root->right = temp;

}

}

void preorder(node \*root) {

if (root != NULL) {

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

preorder(root->left);

preorder(root->right);

}

}

void inorder(node \*root) {

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

void postorder(node \*root) {

if (root != NULL) {

postorder(root->left);

postorder(root->right);

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

}

}

int main() {

char ch;

node \*root = NULL, \*temp;

printf("Do you want to enter data? ");

printf("\Y/N:");

scanf("%c", &ch); // Initial check to start the loop

while (ch == 'y' || ch == 'Y') {

temp = create();

if (root == NULL)

root = temp;

else

insert(root, temp);

printf("\nEnter more data?\nY/N:");

getchar(); // To consume the newline character left by scanf

scanf("%c", &ch); // Read the user's input

}

printf("\nPreorder Traversal: ");

preorder(root);

printf("\nInorder Traversal: ");

inorder(root);

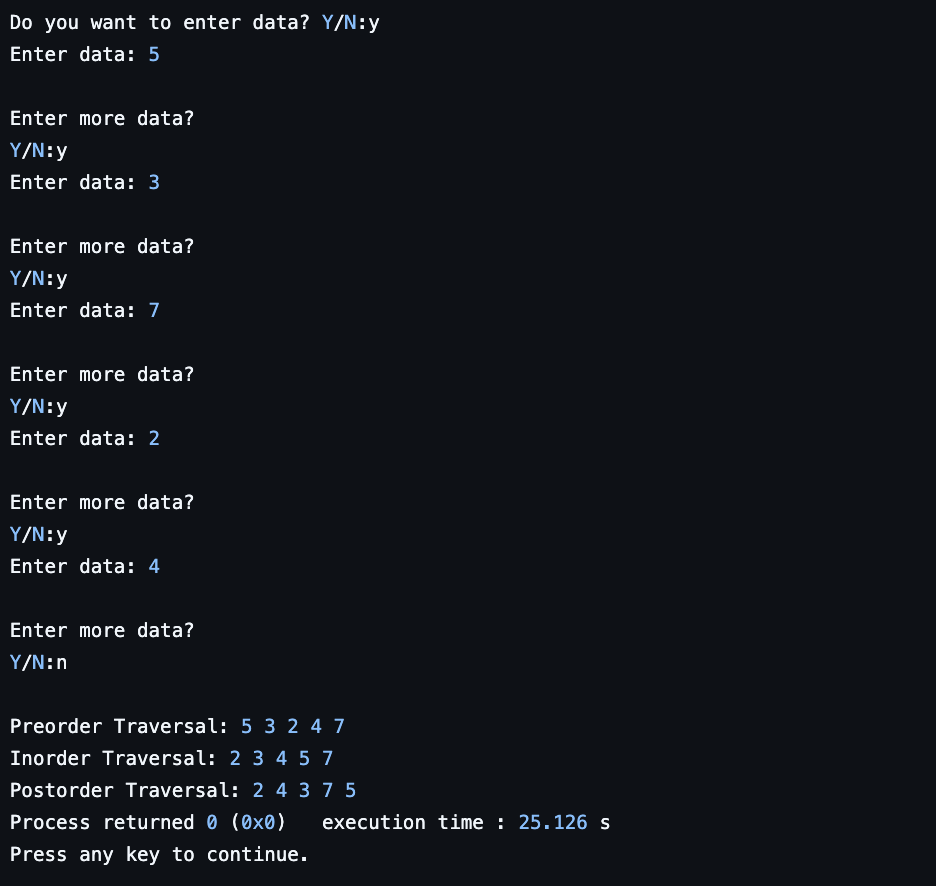
printf("\nPostorder Traversal: ");

postorder(root);

return 0;

}

**Output:**



**2) LEET CODE: (hasPathSum)**

bool hasPathSum(struct TreeNode\* root, int targetSum) {

if (root == NULL) {

return false;

}

targetSum -= root->val;

if (root->left == NULL && root->right == NULL) {

return targetSum == 0;

}

return hasPathSum(root->left, targetSum) || hasPathSum(root->right, targetSum);

}

**Lab program 9:**

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

#include <stdio.h>

void bfs(int adj[10][10], int n, int source){

int que[10];

int front=0,rear=-1;

int visited[10]={0};

int node;

printf("The nodes visited from %d: ", source);

que[++rear]=source;

visited[source]=1;

printf("%d",source);

while(front<=rear){

int u= que[front++];

for(int v=0; v<n; v++){

if(adj[u][v]==1){

if(visited[v]==0){

printf("%d",v);

visited[v]=1;

que[++rear]=v;

}

}

}

}

printf("\n");

}

int main() {

int n;

int adj[10][10];

int source;

printf("enter number of nodes \n");

scanf("%d",&n);

printf("Enter Adjacency Matrix \n");

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

for(int j=0; j<n; j++){

scanf("%d",&adj[i][j]);

}

}

for(source=0; source<n; source++){

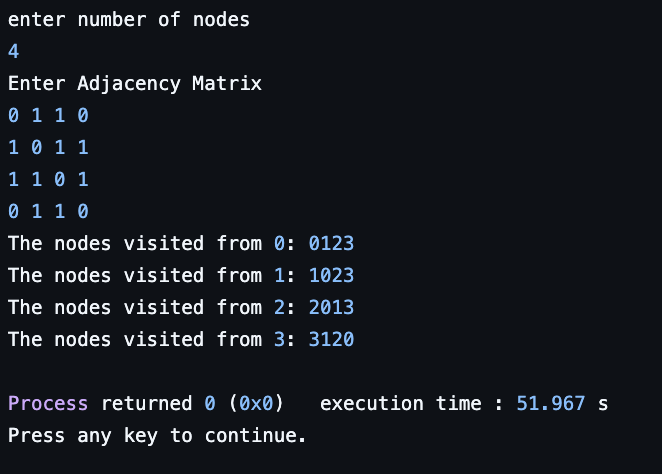
bfs(adj,n,source);

}

return 0;

}

**Output:**



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

#include <stdio.h>

#include <stdlib.h>

#define MAX\_NODES 100

int graph[MAX\_NODES][MAX\_NODES];

int visited[MAX\_NODES];

int stack[MAX\_NODES];

int top = -1;

int nodes;

int DFS(int startNode) {

int visitedCount = 0;

stack[++top] = startNode;

while (top != -1) {

int node = stack[top--];

if (!visited[node]) {

visited[node] = 1;

visitedCount++;

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

if (graph[node][i] == 1 && !visited[i]) {

stack[++top] = i;

}

}

}

}

return visitedCount;

}

int main() {

int edges;

printf("Enter the number of nodes: ");

scanf("%d", &nodes);

printf("Enter the number of edges: ");

scanf("%d", &edges);

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

for (int j = 0; j < nodes; j++) {

graph[i][j] = 0;

}

}

printf("Enter the edges (u v) where u and v are node indices starting from 0:\n");

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

int u, v;

scanf("%d %d", &u, &v);

graph[u][v] = 1;

graph[v][u] = 1;

}

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

visited[i] = 0;

}

int visitedNodes = DFS(0);

if (visitedNodes == nodes) {

printf("The graph is connected.\n");

} else {

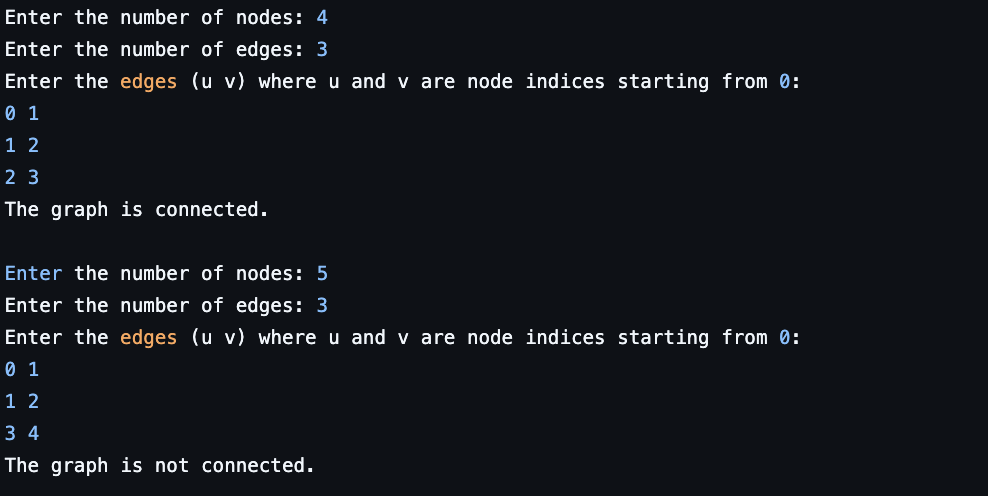
printf("The graph is not connected.\n");

}

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.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TABLE\_SIZE 10

typedef struct Employee {

int key;

char name[50];

char position[50];

} Employee;

Employee hashTable[TABLE\_SIZE];

int isOccupied[TABLE\_SIZE];

void insert(int key, const char\* name, const char\* position) {

int address = key % TABLE\_SIZE;

while (isOccupied[address]) {

address = (address + 1) % TABLE\_SIZE;

}

hashTable[address].key = key;

strcpy(hashTable[address].name, name);

strcpy(hashTable[address].position, position);

isOccupied[address] = 1;

}

Employee\* search(int key) {

int address = key % TABLE\_SIZE;

int startAddress = address;

while (isOccupied[address]) {

if (hashTable[address].key == key) {

return &hashTable[address];

}

address = (address + 1) % TABLE\_SIZE;

if (address == startAddress) {

break;

}

}

return NULL;

}

void displayHashTable() {

for (int i = 0; i < TABLE\_SIZE; i++) {

if (isOccupied[i]) {

printf("[%02d]: (Key: %d, Name: %s, Position: %s)\n", i, hashTable[i].key, hashTable[i].name, hashTable[i].position);

} else {

printf("[%02d]: NULL\n", i);

}

}

}

int main() {

for (int i = 0; i < TABLE\_SIZE; i++) {

isOccupied[i] = 0;

}

int continueInput = 1;

while (continueInput) {

int key;

char name[50], position[50];

printf("\nEnter employee details:\n");

printf("Enter key: ");

scanf("%d", &key);

getchar();

printf("Enter name: ");

fgets(name, sizeof(name), stdin);

name[strcspn(name, "\n")] = '\0';

printf("Enter position: ");

fgets(position, sizeof(position), stdin);

position[strcspn(position, "\n")] = '\0';

insert(key, name, position);

printf("\nDo you want to add another employee? (1 for yes, 0 for no): ");

scanf("%d", &continueInput);

getchar();

}

printf("\nHash Table:\n");

displayHashTable();

int searchKey;

printf("\nEnter the key to search for: ");

scanf("%d", &searchKey);

Employee\* result = search(searchKey);

if (result != NULL) {

printf("\nEmployee found: Key: %d, Name: %s, Position: %s\n", result->key, result->name, result->position);

} else {

printf("\nEmployee with key %d not found.\n", searchKey);

}

return 0;

}

**Output :**



**END OF REPORT**