**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

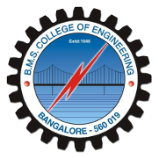
**SHASHANK.S (1BM24CS266)**

**in partial fulfillment for the award of the degree of**

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**August-December 2025**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

**(Affiliated To Visvesvaraya Technological University, Belgaum)**

**Department of Computer Science and Engineering**

****

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by Shashank.S **(1BM24CS266)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2025-2026. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.

**Dr. Namratha 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 |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |

**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. |

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>

#include <stdlib.h>

#define MAX 5

int stack[MAX];

int top = -1;

void push() {

int item;

if (top == MAX - 1) {

printf("Stack Overflow\n");

return;

}

printf("Enter element to push: ");

scanf("%d", &item);

stack[++top] = item;

printf("Element pushed successfully\n");

}

void pop() {

if (top == -1) {

printf("Stack Underflow\n");

return;

}

printf("Popped element: %d\n", stack[top--]);

}

void display() {

int i;

if (top == -1) {

printf("Stack is empty\n");

return;

}

printf("Stack elements:\n");

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

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

}

}

int main() {

int choice;

while (1) {

printf("\n--- STACK MENU ---\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: push(); break;

case 2: pop(); break;

case 3: display(); break;

case 4: exit(0);

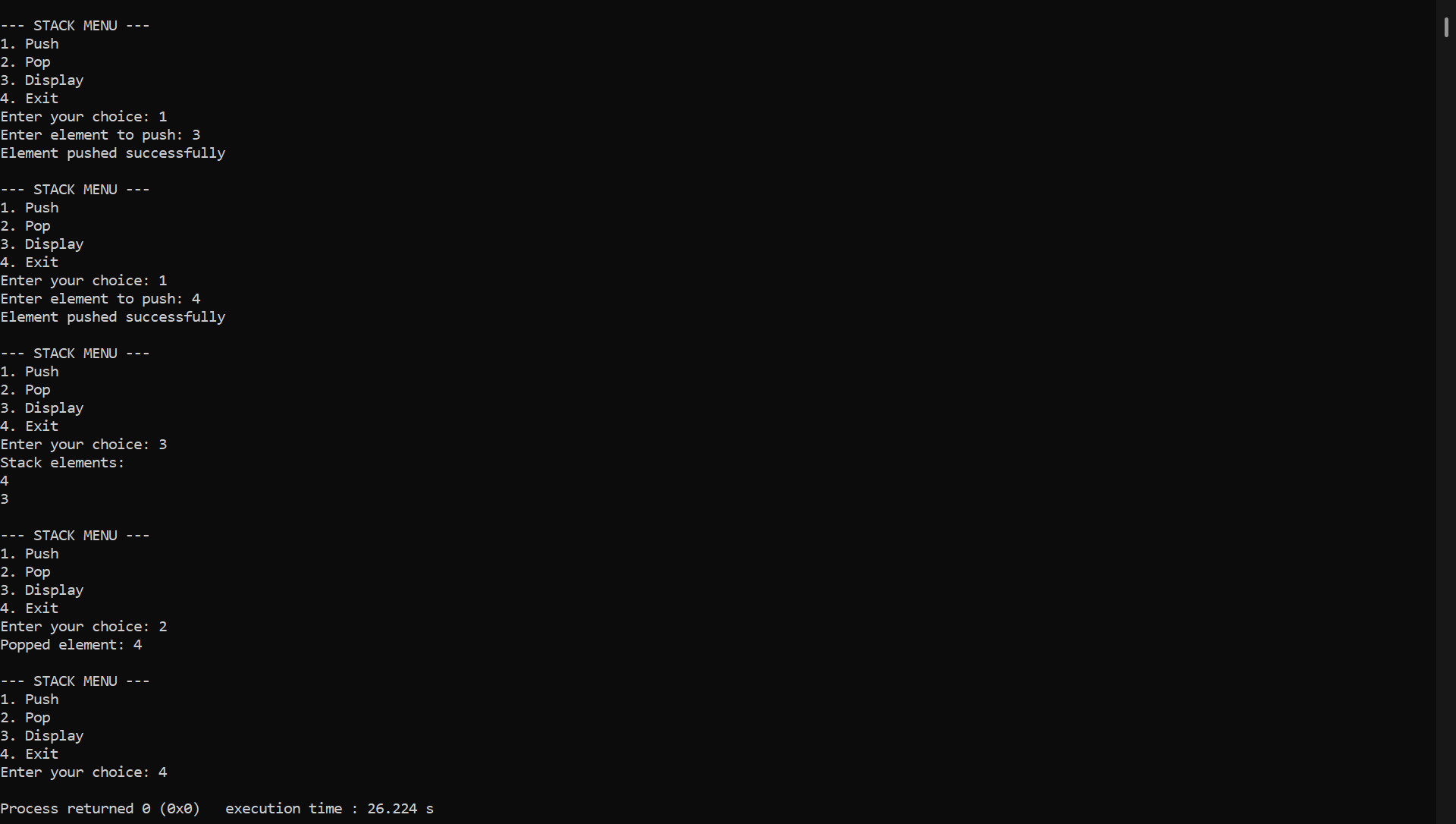
default: printf("Invalid choice\n");

}

}

return 0;

}



**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).**

**PROGRAMN:**

**#include <stdio.h>**

**#include <ctype.h>**

**#define MAX 100**

**char stack[MAX];**

**int top = -1;**

**void push(char x) {**

**stack[++top] = x;**

**}**

**char pop() {**

**return stack[top--];**

**}**

**int priority(char x) {**

**if (x == '+' || x == '-')**

**return 1;**

**if (x == '\*' || x == '/')**

**return 2;**

**return 0;**

**}**

**int main() {**

**char infix[MAX], postfix[MAX];**

**int i = 0, j = 0;**

**char ch;**

**printf("Enter infix expression: ");**

**scanf("%s", infix);**

**while ((ch = infix[i++]) != '\0') {**

**if (isalnum(ch)) {**

**postfix[j++] = ch;**

**}**

**else if (ch == '(') {**

**push(ch);**

**}**

**else if (ch == ')') {**

**while (stack[top] != '(')**

**postfix[j++] = pop();**

**pop(); // remove '('**

**}**

**else {**

**while (top != -1 && priority(stack[top]) >= priority(ch))**

**postfix[j++] = pop();**

**push(ch);**

**}**

**}**

**while (top != -1)**

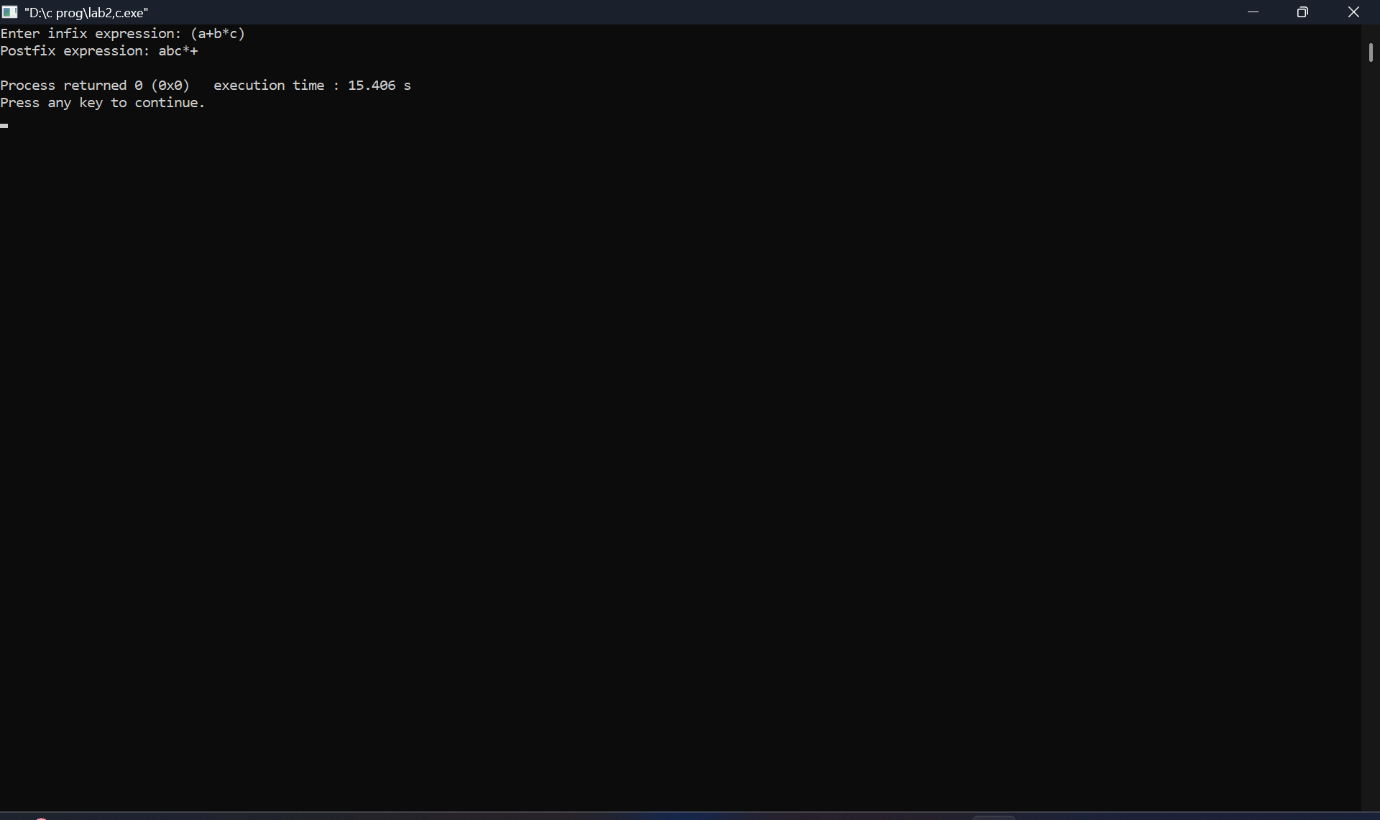
**postfix[j++] = pop();**

**postfix[j] = '\0';**

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

**return 0;**

**}**

****

**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**

**PROGRAMN:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 5**

**int queue[MAX];**

**int front = -1, rear = -1;**

**void insert() {**

**int item;**

**if (rear == MAX - 1) {**

**printf("Queue Overflow\n");**

**return;**

**}**

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

**scanf("%d", &item);**

**if (front == -1)**

**front = 0;**

**queue[++rear] = item;**

**printf("Element inserted successfully\n");**

**}**

**void delete() {**

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

**printf("Queue Empty\n");**

**return;**

**}**

**printf("Deleted element: %d\n", queue[front++]);**

**}**

**void display() {**

**int i;**

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

**printf("Queue Empty\n");**

**return;**

**}**

**printf("Queue elements: ");**

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

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

**}**

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

**}**

**int main() {**

**int choice;**

**while (1) {**

**printf("\n===== QUEUE MENU =====\n");**

**printf("1. Insert\n");**

**printf("2. Delete\n");**

**printf("3. Display\n");**

**printf("4. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1: insert(); break;**

**case 2: delete(); break;**

**case 3: display(); break;**

**case 4: exit(0);**

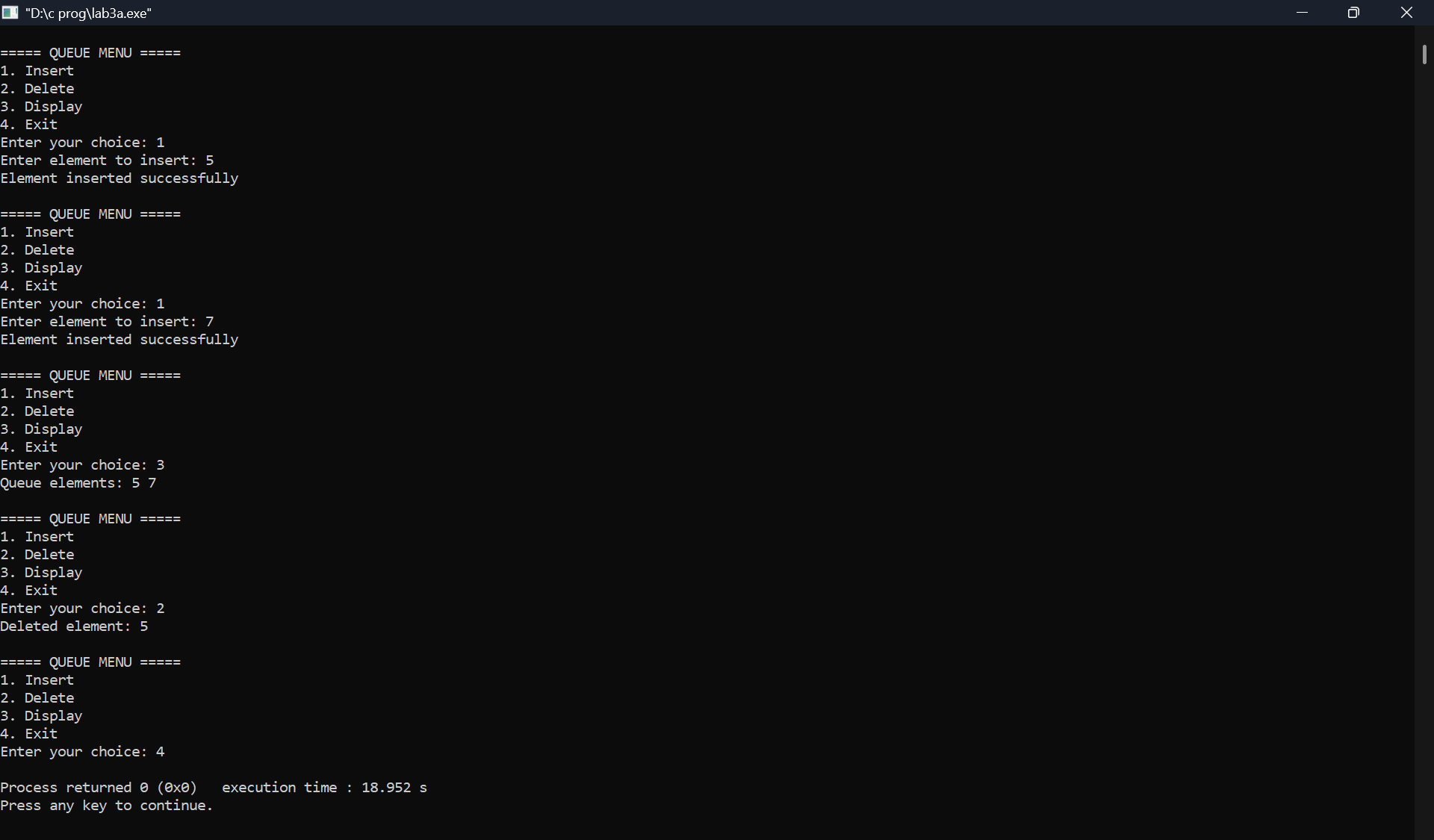
**default: printf("Invalid choice\n");**

**}**

**}**

**return 0;**

**}**

****

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

**PROGRAMN:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 5**

**int cq[MAX];**

**int front = -1, rear = -1;**

**void insert() {**

**int item;**

**if ((front == 0 && rear == MAX - 1) || (front == rear + 1)) {**

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

**return;**

**}**

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

**scanf("%d", &item);**

**if (front == -1) {**

**front = rear = 0;**

**} else if (rear == MAX - 1) {**

**rear = 0;**

**} else {**

**rear++;**

**}**

**cq[rear] = item;**

**printf("Element inserted successfully\n");**

**}**

**void delete() {**

**if (front == -1) {**

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

**return;**

**}**

**printf("Deleted element: %d\n", cq[front]);**

**if (front == rear) {**

**front = rear = -1;**

**} else if (front == MAX - 1) {**

**front = 0;**

**} else {**

**front++;**

**}**

**}**

**void display() {**

**int i;**

**if (front == -1) {**

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

**return;**

**}**

**printf("Circular Queue elements: ");**

**i = front;**

**while (1) {**

**printf("%d ", cq[i]);**

**if (i == rear)**

**break;**

**i = (i + 1) % MAX;**

**}**

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

**}**

**int main() {**

**int choice;**

**while (1) {**

**printf("\n===== CIRCULAR QUEUE MENU =====\n");**

**printf("1. Insert\n");**

**printf("2. Delete\n");**

**printf("3. Display\n");**

**printf("4. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1: insert(); break;**

**case 2: delete(); break;**

**case 3: display(); break;**

**case 4: exit(0);**

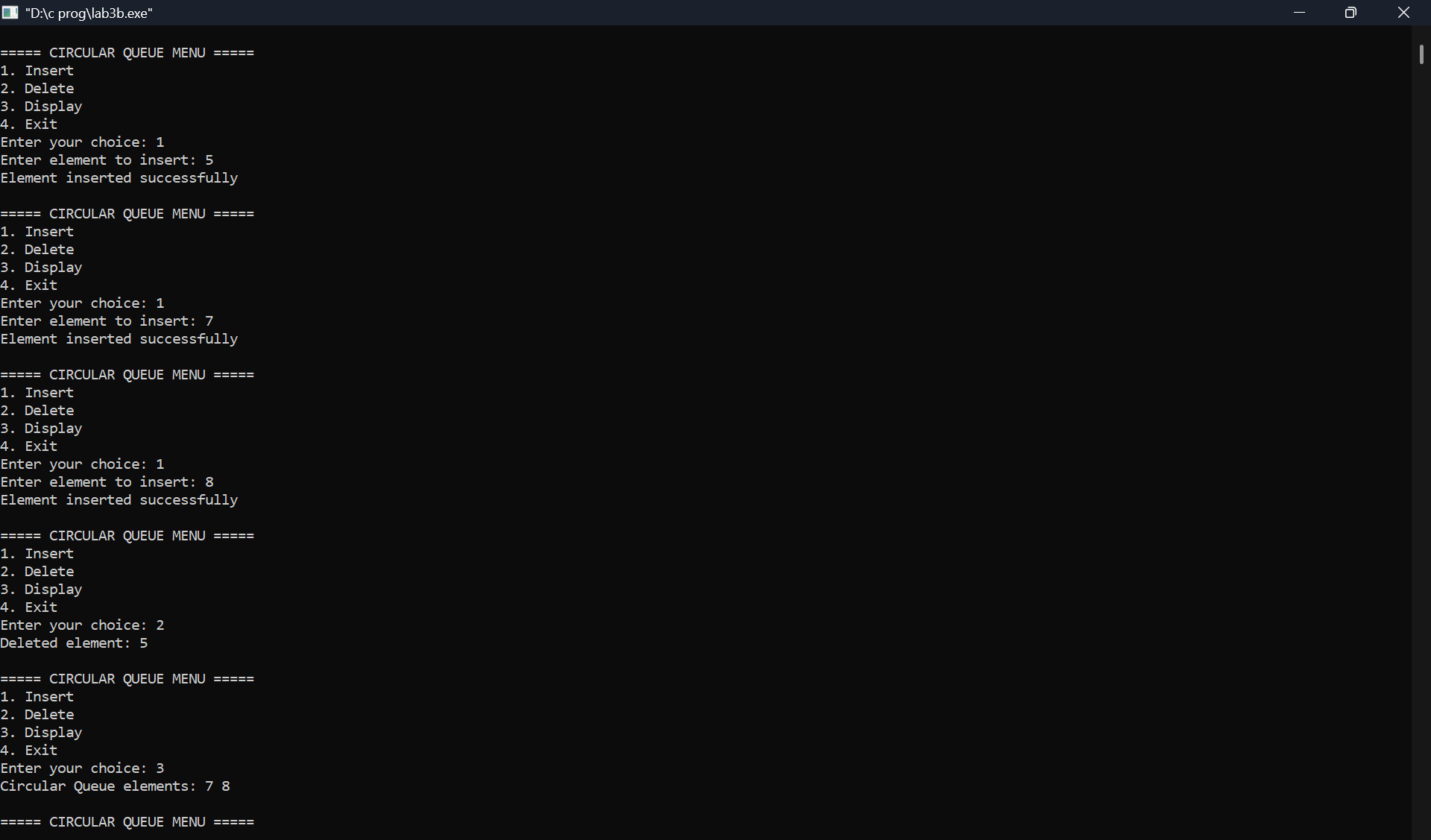
**default: printf("Invalid choice\n");**

**}**

**}**

**return 0;**

**}**

****

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

**PROGRAMN:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*next;**

**};**

**struct Node \*head = NULL;**

**void create() {**

**int item;**

**struct Node \*temp, \*newNode;**

**printf("Enter elements (-999 to stop): ");**

**scanf("%d", &item);**

**while (item != -999) {**

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

**newNode->data = item;**

**newNode->next = NULL;**

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

**head = newNode;**

**} else {**

**temp = head;**

**while (temp->next != NULL)**

**temp = temp->next;**

**temp->next = newNode;**

**}**

**scanf("%d", &item);**

**}**

**}**

**void insert\_begin() {**

**int item;**

**struct Node \*newNode;**

**printf("Enter element to insert at beginning: ");**

**scanf("%d", &item);**

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

**newNode->data = item;**

**newNode->next = head;**

**head = newNode;**

**}**

**void insert\_pos() {**

**int item, pos, i;**

**struct Node \*newNode, \*temp;**

**printf("Enter element and position: ");**

**scanf("%d %d", &item, &pos);**

**if (pos == 1) {**

**insert\_begin();**

**return;**

**}**

**temp = head;**

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

**temp = temp->next;**

**if (temp == NULL) {**

**printf("Invalid position\n");**

**return;**

**}**

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

**newNode->data = item;**

**newNode->next = temp->next;**

**temp->next = newNode;**

**}**

**void insert\_end() {**

**int item;**

**struct Node \*newNode, \*temp;**

**printf("Enter element to insert at end: ");**

**scanf("%d", &item);**

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

**newNode->data = item;**

**newNode->next = NULL;**

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

**head = newNode;**

**return;**

**}**

**temp = head;**

**while (temp->next != NULL)**

**temp = temp->next;**

**temp->next = newNode;**

**}**

**void display() {**

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

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

**printf("List is empty\n");**

**return;**

**}**

**printf("Linked List: ");**

**while (temp != NULL) {**

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

**temp = temp->next;**

**}**

**printf("NULL\n");**

**}**

**int main() {**

**int choice;**

**while (1) {**

**printf("\n===== SINGLY LINKED LIST MENU =====\n");**

**printf("1. Create List\n");**

**printf("2. Insert at Beginning\n");**

**printf("3. Insert at Position\n");**

**printf("4. Insert at End\n");**

**printf("5. Display\n");**

**printf("6. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1: create(); break;**

**case 2: insert\_begin(); break;**

**case 3: insert\_pos(); break;**

**case 4: insert\_end(); break;**

**case 5: display(); break;**

**case 6: exit(0);**

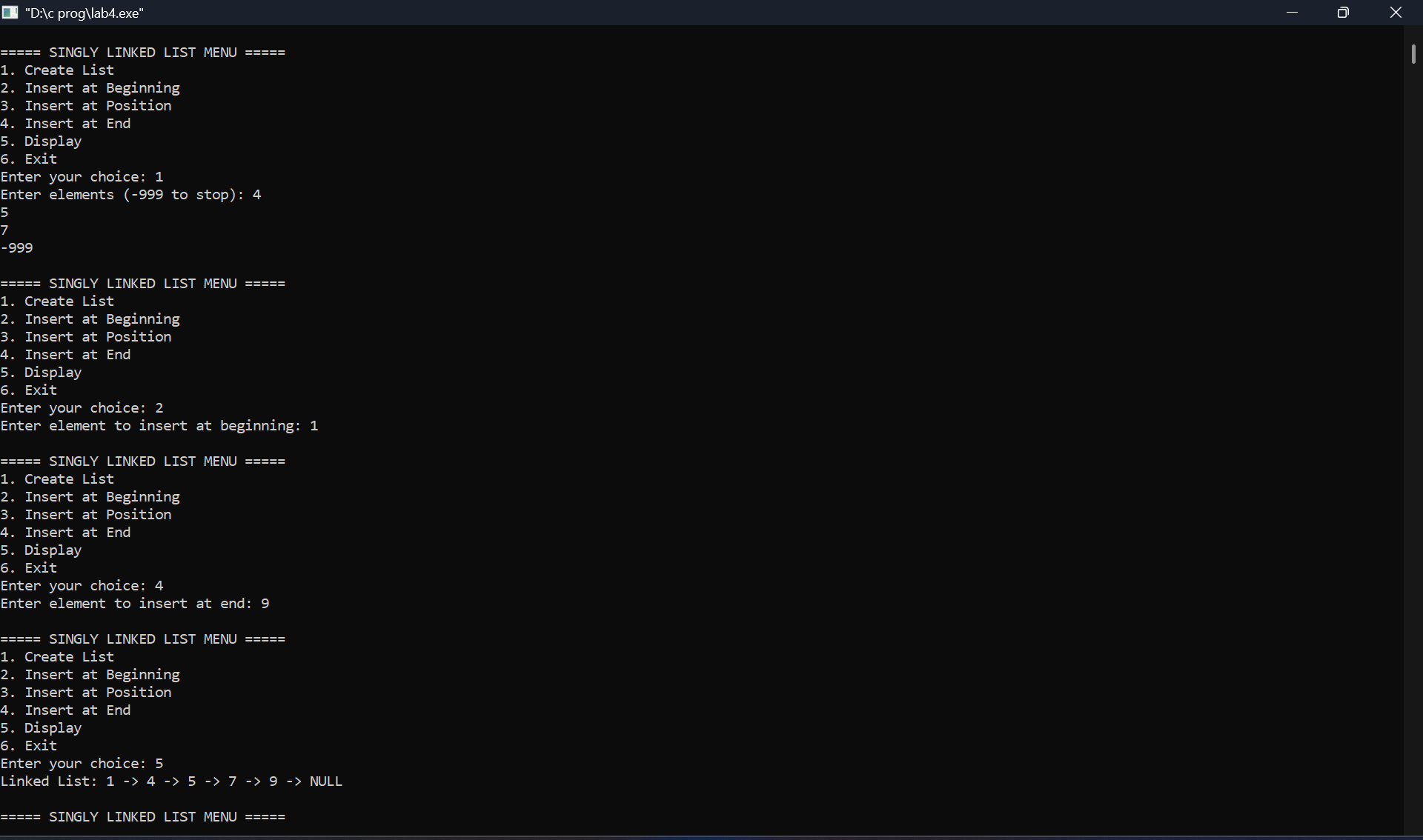
**default: printf("Invalid choice\n");**

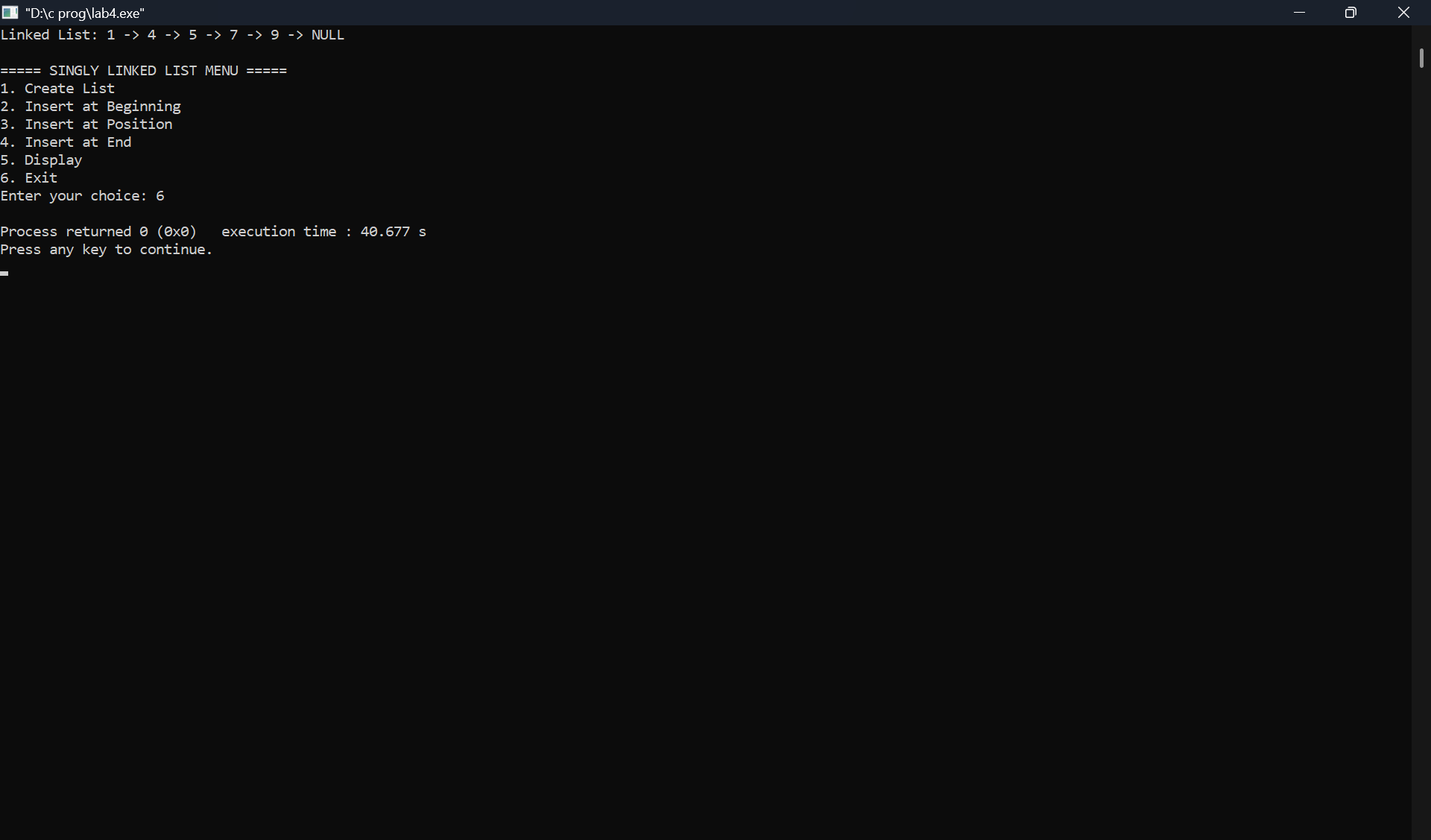
**}**

**}**

**return 0;**

**}**

****

****

**4] b]Program - Leetcode platform Given the head of a singly linked list, return *the middle node of the linked list*.**

**If there are two middle nodes, return the second middle node.**

**Function:**

**struct ListNode\* middleNode(struct ListNode\* head) {**

**struct ListNode \*slow = head, \*fast = head;**

**while (fast != NULL && fast->next != NULL) {**

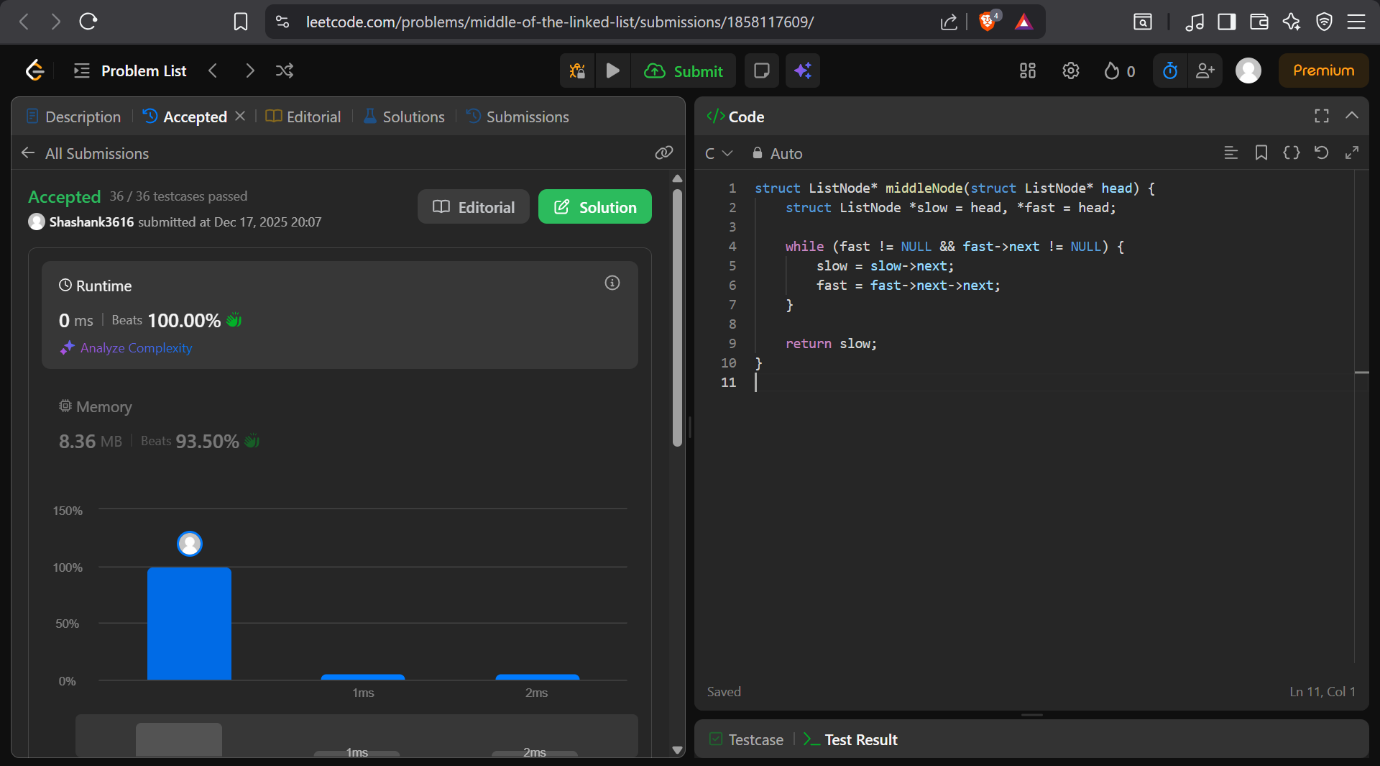
**slow = slow->next;**

**fast = fast->next->next;**

**}**

**return slow;**

**}**

****

**5] a]WAP to Implement Singly Linked List with following operations**

**a) Create a linked list.**

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

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

**PROGRAMN:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*next;**

**};**

**struct Node \*head = NULL;**

**void create() {**

**int item;**

**struct Node \*newNode, \*temp;**

**printf("Enter elements (-999 to stop): ");**

**scanf("%d", &item);**

**while (item != -999) {**

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

**newNode->data = item;**

**newNode->next = NULL;**

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

**head = newNode;**

**} else {**

**temp = head;**

**while (temp->next != NULL)**

**temp = temp->next;**

**temp->next = newNode;**

**}**

**scanf("%d", &item);**

**}**

**}**

**void delete\_first() {**

**struct Node \*temp;**

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

**printf("List is empty\n");**

**return;**

**}**

**temp = head;**

**head = head->next;**

**printf("Deleted element: %d\n", temp->data);**

**free(temp);**

**}**

**void delete\_specified() {**

**int key;**

**struct Node \*temp, \*prev;**

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

**printf("List is empty\n");**

**return;**

**}**

**printf("Enter element to delete: ");**

**scanf("%d", &key);**

**if (head->data == key) {**

**delete\_first();**

**return;**

**}**

**prev = head;**

**temp = head->next;**

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

**prev = temp;**

**temp = temp->next;**

**}**

**if (temp == NULL) {**

**printf("Element not found\n");**

**return;**

**}**

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

**printf("Deleted element: %d\n", temp->data);**

**free(temp);**

**}**

**void delete\_last() {**

**struct Node \*temp, \*prev;**

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

**printf("List is empty\n");**

**return;**

**}**

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

**printf("Deleted element: %d\n", head->data);**

**free(head);**

**head = NULL;**

**return;**

**}**

**prev = head;**

**temp = head->next;**

**while (temp->next != NULL) {**

**prev = temp;**

**temp = temp->next;**

**}**

**prev->next = NULL;**

**printf("Deleted element: %d\n", temp->data);**

**free(temp);**

**}**

**void display() {**

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

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

**printf("List is empty\n");**

**return;**

**}**

**printf("Linked List: ");**

**while (temp != NULL) {**

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

**temp = temp->next;**

**}**

**printf("NULL\n");**

**}**

**int main() {**

**int choice;**

**while (1) {**

**printf("\n===== SINGLY LINKED LIST MENU =====\n");**

**printf("1. Create List\n");**

**printf("2. Delete First\n");**

**printf("3. Delete Specified Element\n");**

**printf("4. Delete Last\n");**

**printf("5. Display\n");**

**printf("6. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1: create(); break;**

**case 2: delete\_first(); break;**

**case 3: delete\_specified(); break;**

**case 4: delete\_last(); break;**

**case 5: display(); break;**

**case 6: exit(0);**

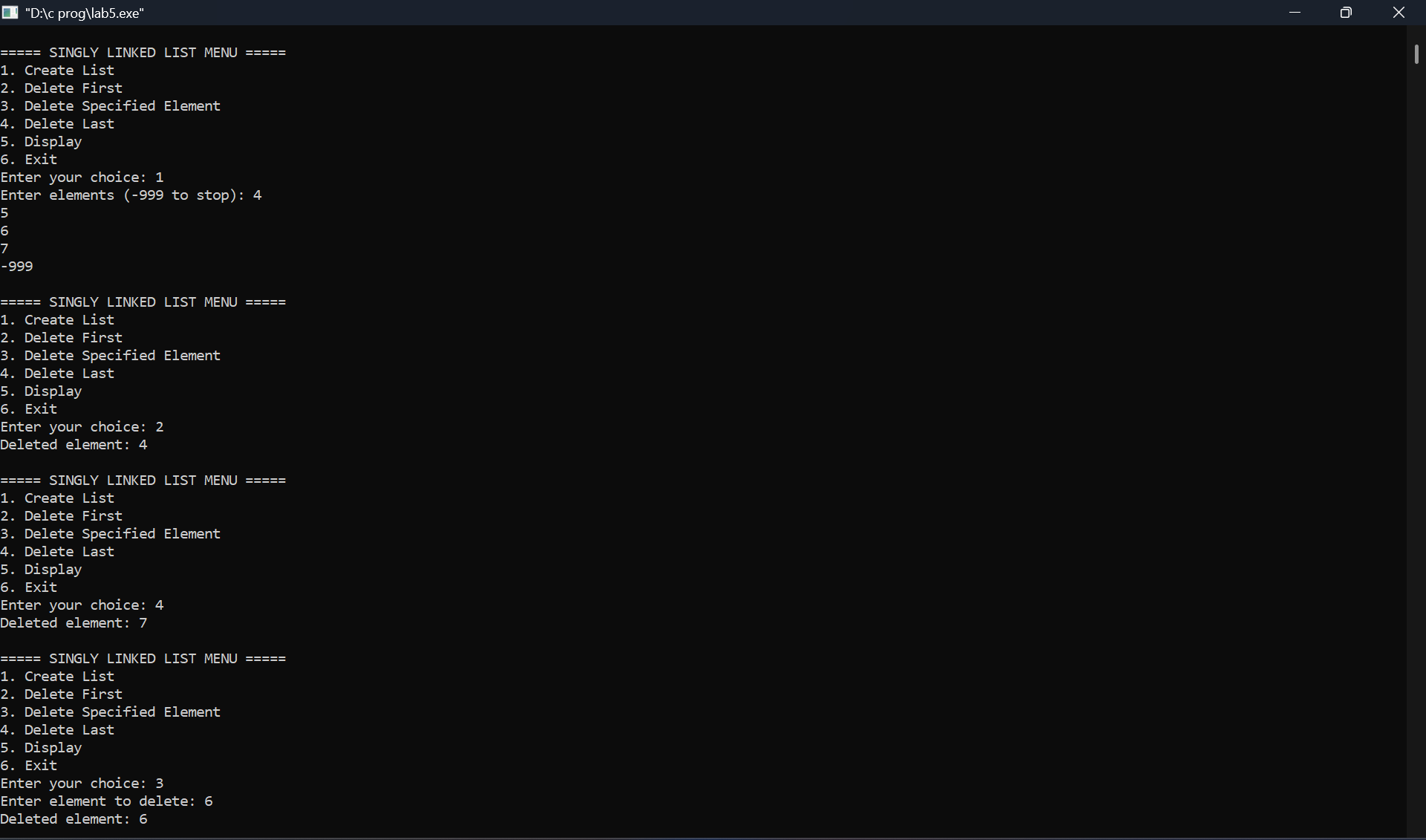
**default: printf("Invalid choice\n");**

**}**

**}**

**return 0;**

**}**

****

****

**5] b] Program - Leetcode platform**

**Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.**

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

**struct ListNode \*curr = head, \*prev = NULL;**

**while (curr != NULL) {**

**if (curr->val == val) {**

**if (prev == NULL) {**

**head = curr->next;**

**free(curr);**

**curr = head;**

**} else {**

**prev->next = curr->next;**

**free(curr);**

**curr = prev->next;**

**}**

**} else {**

**prev = curr;**

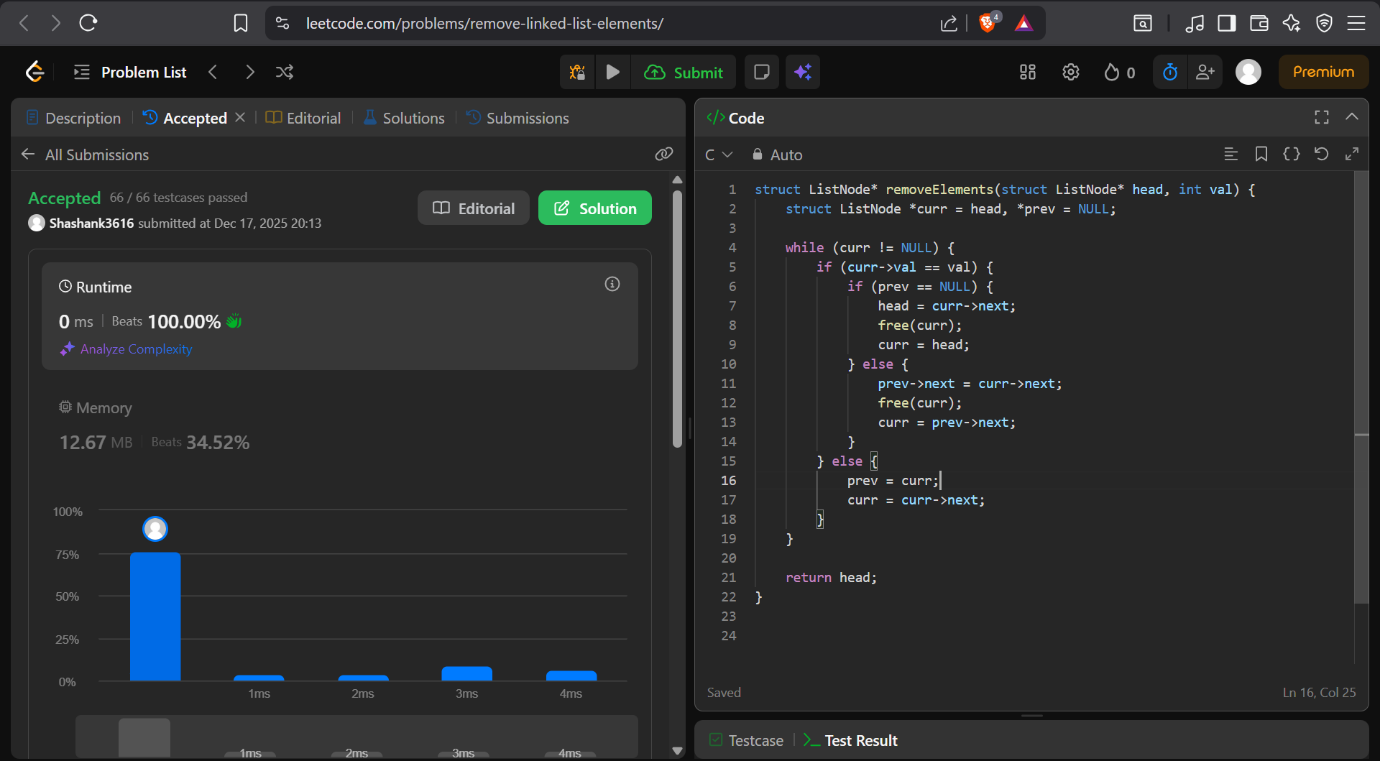
**curr = curr->next;**

**}**

**}**

**return head;**

**}**

****

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

**PROGRAMN:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*next;**

**};**

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

**/\* Create a linked list \*/**

**struct Node\* create() {**

**int item;**

**struct Node \*head = NULL, \*temp, \*newNode;**

**printf("Enter elements (-999 to stop): ");**

**scanf("%d", &item);**

**while (item != -999) {**

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

**newNode->data = item;**

**newNode->next = NULL;**

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

**head = newNode;**

**} else {**

**temp = head;**

**while (temp->next != NULL)**

**temp = temp->next;**

**temp->next = newNode;**

**}**

**scanf("%d", &item);**

**}**

**return head;**

**}**

**/\* Display list \*/**

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

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

**printf("List is empty\n");**

**return;**

**}**

**while (head != NULL) {**

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

**head = head->next;**

**}**

**printf("NULL\n");**

**}**

**/\* Sort linked list \*/**

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

**struct Node \*i, \*j;**

**int temp;**

**if (head == NULL) return;**

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

**for (j = i->next; j != NULL; j = j->next) {**

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

**temp = i->data;**

**i->data = j->data;**

**j->data = temp;**

**}**

**}**

**}**

**}**

**/\* Reverse linked list \*/**

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

**struct Node \*prev = NULL, \*curr = head, \*next = NULL;**

**while (curr != NULL) {**

**next = curr->next;**

**curr->next = prev;**

**prev = curr;**

**curr = next;**

**}**

**return prev;**

**}**

**/\* Concatenate two lists \*/**

**struct Node\* concatenate(struct Node \*head1, struct Node \*head2) {**

**struct Node \*temp;**

**if (head1 == NULL) return head2;**

**if (head2 == NULL) return head1;**

**temp = head1;**

**while (temp->next != NULL)**

**temp = temp->next;**

**temp->next = head2;**

**return head1;**

**}**

**int main() {**

**int choice;**

**while (1) {**

**printf("\n===== SLL MENU =====\n");**

**printf("1. Create List 1\n");**

**printf("2. Create List 2\n");**

**printf("3. Display List 1\n");**

**printf("4. Sort List 1\n");**

**printf("5. Reverse List 1\n");**

**printf("6. Concatenate List 1 & List 2\n");**

**printf("7. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1: head1 = create(); break;**

**case 2: head2 = create(); break;**

**case 3: display(head1); break;**

**case 4:**

**sortList(head1);**

**printf("List sorted\n");**

**break;**

**case 5:**

**head1 = reverseList(head1);**

**printf("List reversed\n");**

**break;**

**case 6:**

**head1 = concatenate(head1, head2);**

**printf("Lists concatenated\n");**

**break;**

**case 7:**

**exit(0);**

**default:**

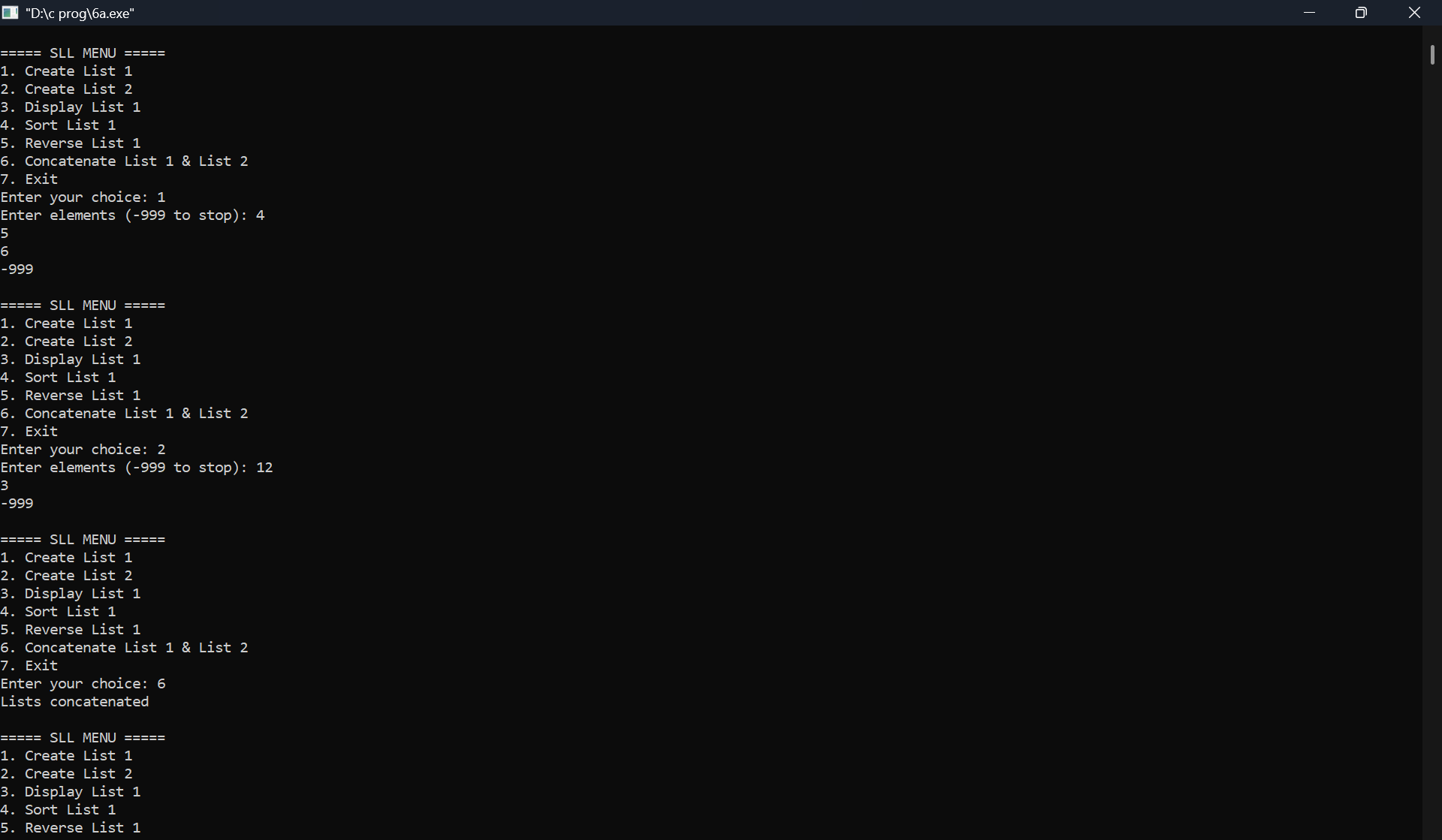
**printf("Invalid choice\n");**

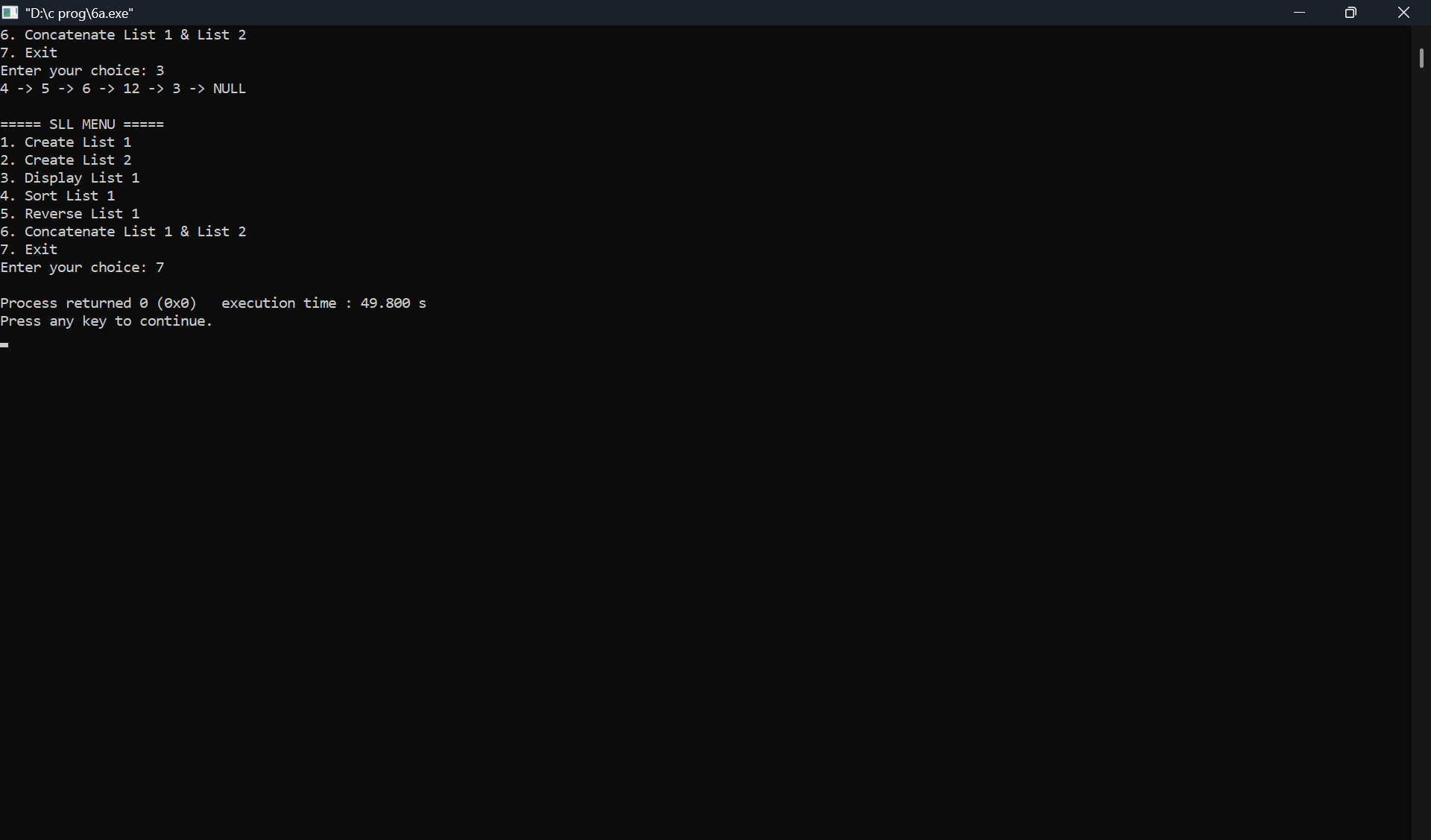
**}**

**}**

**return 0;**

**}**

****

****

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

**PROGRAMN:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*next;**

**};**

**struct Node \*head = NULL;**

**/\* Insert at beginning (Stack Push) \*/**

**void push(int item) {**

**struct Node \*newNode;**

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

**newNode->data = item;**

**newNode->next = head;**

**head = newNode;**

**printf("Element pushed\n");**

**}**

**/\* Delete from beginning (Stack Pop / Queue Dequeue) \*/**

**void pop\_dequeue() {**

**struct Node \*temp;**

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

**printf("List is empty\n");**

**return;**

**}**

**temp = head;**

**printf("Deleted element: %d\n", temp->data);**

**head = head->next;**

**free(temp);**

**}**

**/\* Insert at end (Queue Enqueue) \*/**

**void enqueue(int item) {**

**struct Node \*newNode, \*temp;**

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

**newNode->data = item;**

**newNode->next = NULL;**

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

**head = newNode;**

**return;**

**}**

**temp = head;**

**while (temp->next != NULL)**

**temp = temp->next;**

**temp->next = newNode;**

**printf("Element enqueued\n");**

**}**

**/\* Display list \*/**

**void display() {**

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

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

**printf("List is empty\n");**

**return;**

**}**

**printf("List: ");**

**while (temp != NULL) {**

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

**temp = temp->next;**

**}**

**printf("NULL\n");**

**}**

**int main() {**

**int choice, item;**

**while (1) {**

**printf("\n===== STACK & QUEUE MENU =====\n");**

**printf("1. Stack Push\n");**

**printf("2. Stack Pop\n");**

**printf("3. Queue Enqueue\n");**

**printf("4. Queue Dequeue\n");**

**printf("5. Display\n");**

**printf("6. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**printf("Enter value: ");**

**scanf("%d", &item);**

**push(item);**

**break;**

**case 2:**

**pop\_dequeue();**

**break;**

**case 3:**

**printf("Enter value: ");**

**scanf("%d", &item);**

**enqueue(item);**

**break;**

**case 4:**

**pop\_dequeue();**

**break;**

**case 5:**

**display();**

**break;**

**case 6:**

**exit(0);**

**default:**

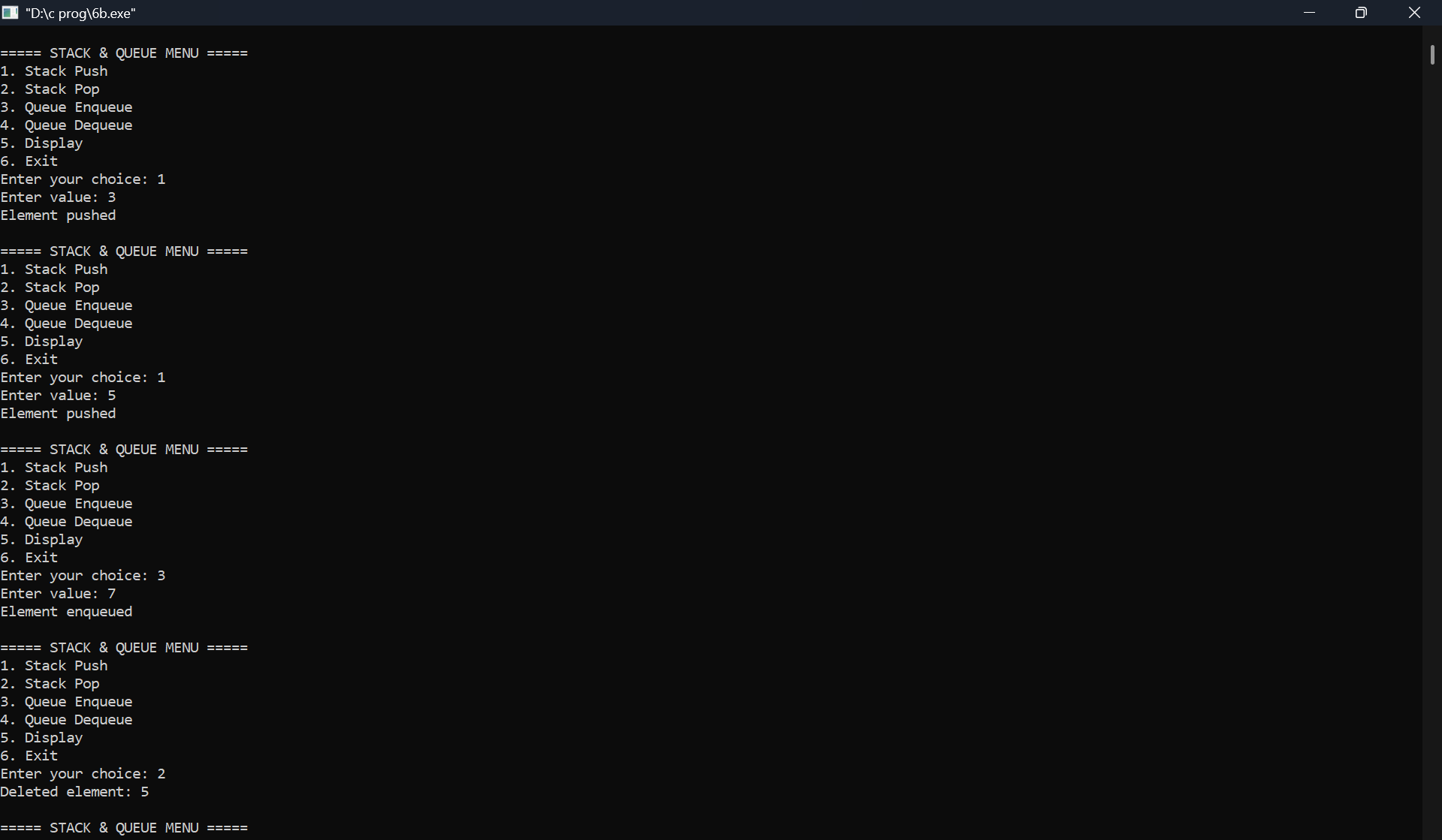
**printf("Invalid choice\n");**

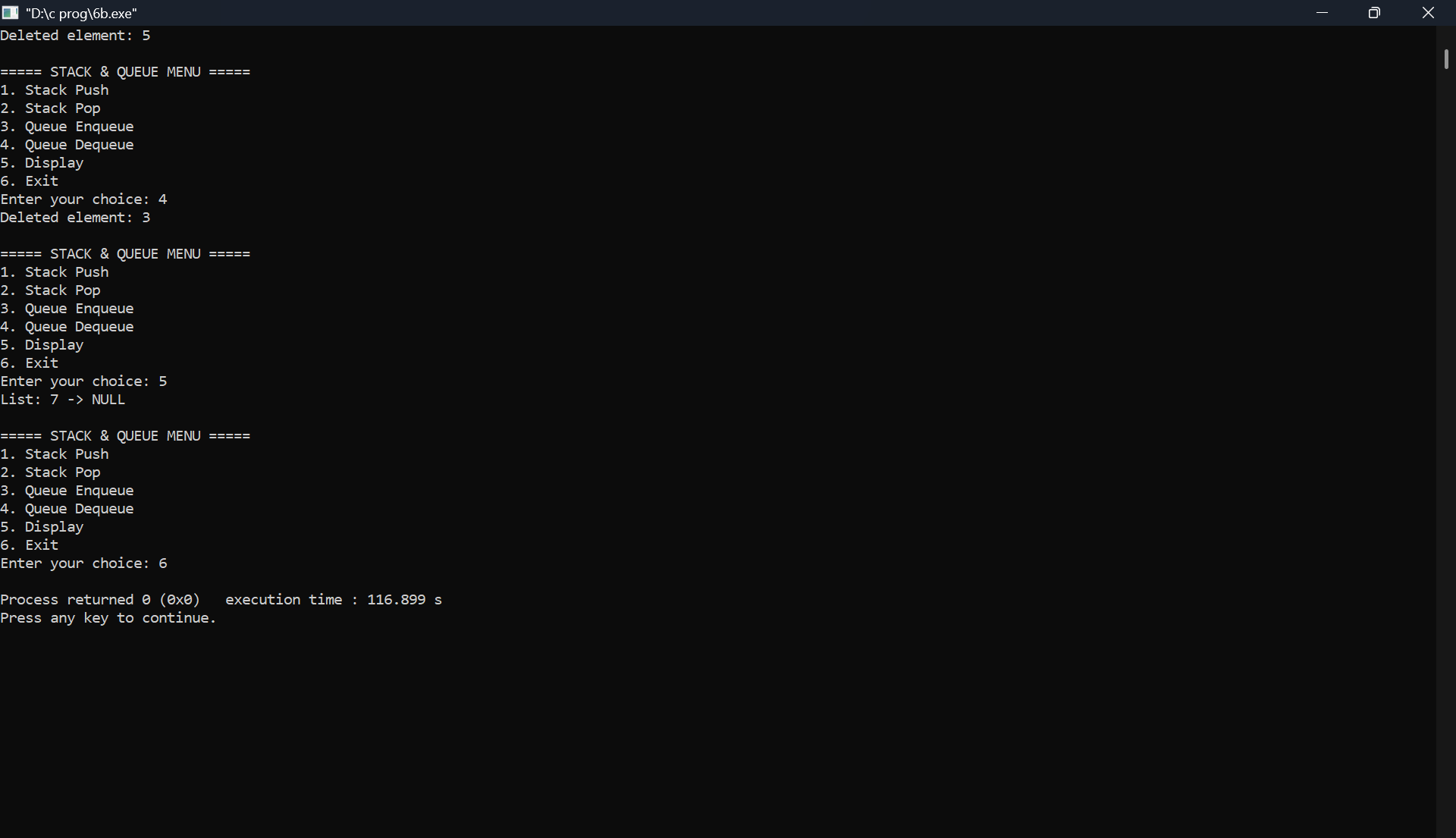
**}**

**}**

**return 0;**

**}**

****

****

**7] a]** **WAP to Implement doubly link list with primitive operations**

**a) Create a doubly linked list.**

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

**c) Delete the node based on a specific value**

**d) Display the contents of the list**

**PROGRAMN:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node {**

**int info;**

**struct node \*prev;**

**struct node \*next;**

**};**

**struct node \*start = NULL;**

**/\* Create doubly linked list \*/**

**void create() {**

**int item;**

**struct node \*newnode, \*temp;**

**printf("Enter elements (-999 to stop): ");**

**scanf("%d", &item);**

**while (item != -999) {**

**newnode = (struct node \*)malloc(sizeof(struct node));**

**newnode->info = item;**

**newnode->prev = newnode->next = NULL;**

**if (start == NULL) {**

**start = newnode;**

**} else {**

**temp = start;**

**while (temp->next != NULL)**

**temp = temp->next;**

**temp->next = newnode;**

**newnode->prev = temp;**

**}**

**scanf("%d", &item);**

**}**

**}**

**/\* Insert a node to the left of a given value \*/**

**void insert\_left() {**

**int value, key;**

**struct node \*temp, \*newnode;**

**if (start == NULL) {**

**printf("List is empty\n");**

**return;**

**}**

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

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

**printf("Enter key value (insert to its left): ");**

**scanf("%d", &key);**

**temp = start;**

**while (temp != NULL && temp->info != key)**

**temp = temp->next;**

**if (temp == NULL) {**

**printf("Key not found\n");**

**return;**

**}**

**newnode = (struct node \*)malloc(sizeof(struct node));**

**newnode->info = value;**

**newnode->next = temp;**

**newnode->prev = temp->prev;**

**if (temp->prev != NULL)**

**temp->prev->next = newnode;**

**else**

**start = newnode;**

**temp->prev = newnode;**

**printf("Node inserted successfully\n");**

**}**

**/\* Delete a node based on a specific value \*/**

**void delete\_value() {**

**int key;**

**struct node \*temp;**

**if (start == NULL) {**

**printf("List is empty\n");**

**return;**

**}**

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

**scanf("%d", &key);**

**temp = start;**

**while (temp != NULL && temp->info != key)**

**temp = temp->next;**

**if (temp == NULL) {**

**printf("Value not found\n");**

**return;**

**}**

**if (temp->prev != NULL)**

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

**else**

**start = temp->next;**

**if (temp->next != NULL)**

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

**printf("Deleted element: %d\n", temp->info);**

**free(temp);**

**}**

**/\* Display the list \*/**

**void display() {**

**struct node \*temp = start;**

**if (start == NULL) {**

**printf("List is empty\n");**

**return;**

**}**

**printf("Doubly Linked List: ");**

**while (temp != NULL) {**

**printf("%d <-> ", temp->info);**

**temp = temp->next;**

**}**

**printf("NULL\n");**

**}**

**int main() {**

**int choice;**

**while (1) {**

**printf("\n===== DOUBLY LINKED LIST MENU =====\n");**

**printf("1. Create List\n");**

**printf("2. Insert Left of a Node\n");**

**printf("3. Delete by Value\n");**

**printf("4. Display\n");**

**printf("5. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1: create(); break;**

**case 2: insert\_left(); break;**

**case 3: delete\_value(); break;**

**case 4: display(); break;**

**case 5: exit(0);**

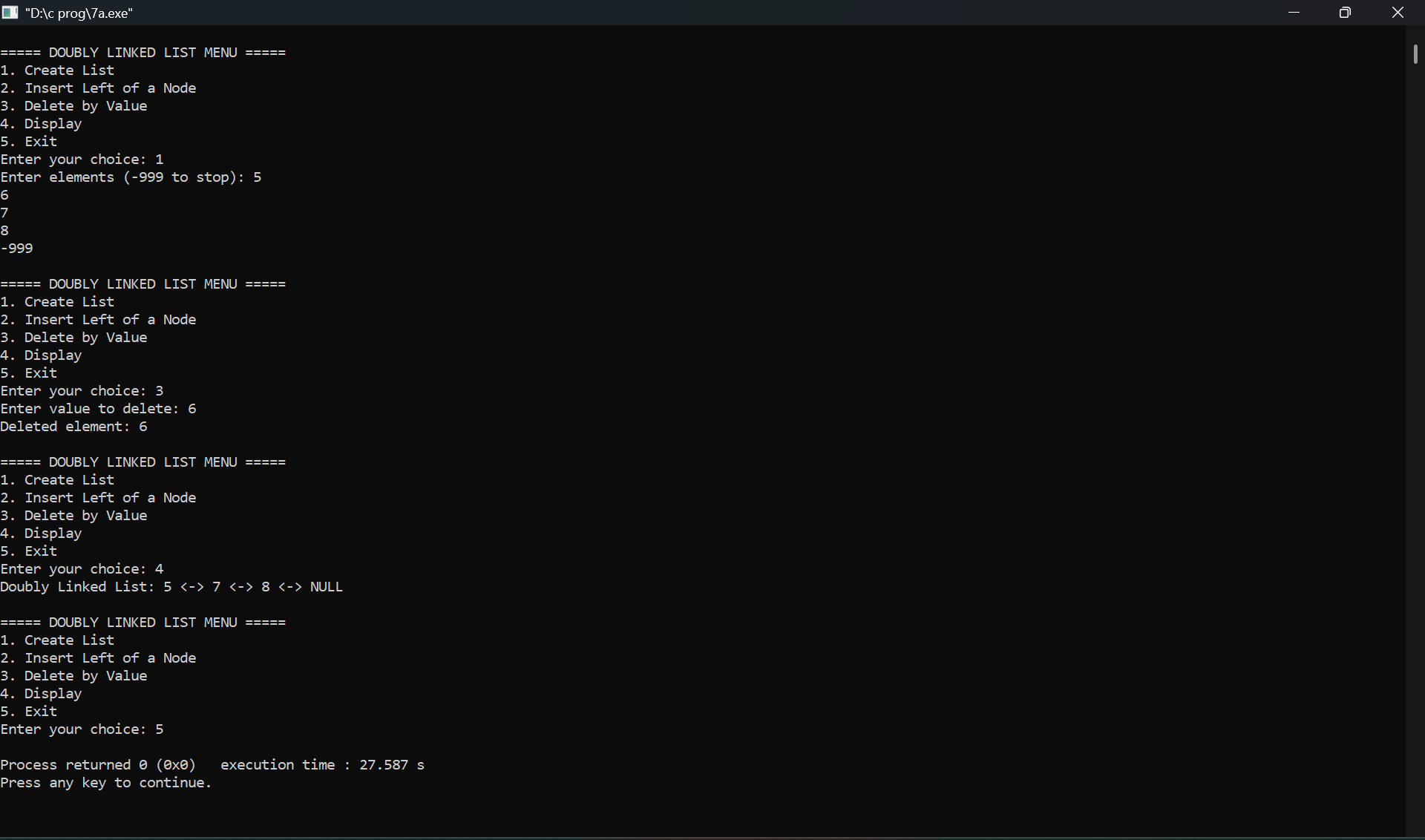
**default: printf("Invalid choice\n");**

**}**

**}**

**return 0;**

**}**

****

**7 b] LEET CODE - Given head, the head of a linked list, determine if the linked list has a cycle in it.**

**There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.**

**FUNCTION:**

**bool hasCycle(struct ListNode \*head) {**

**struct ListNode \*slow = head;**

**struct ListNode \*fast = head;**

**while (fast != NULL && fast->next != NULL) {**

**slow = slow->next;**

**fast = fast->next->next;**

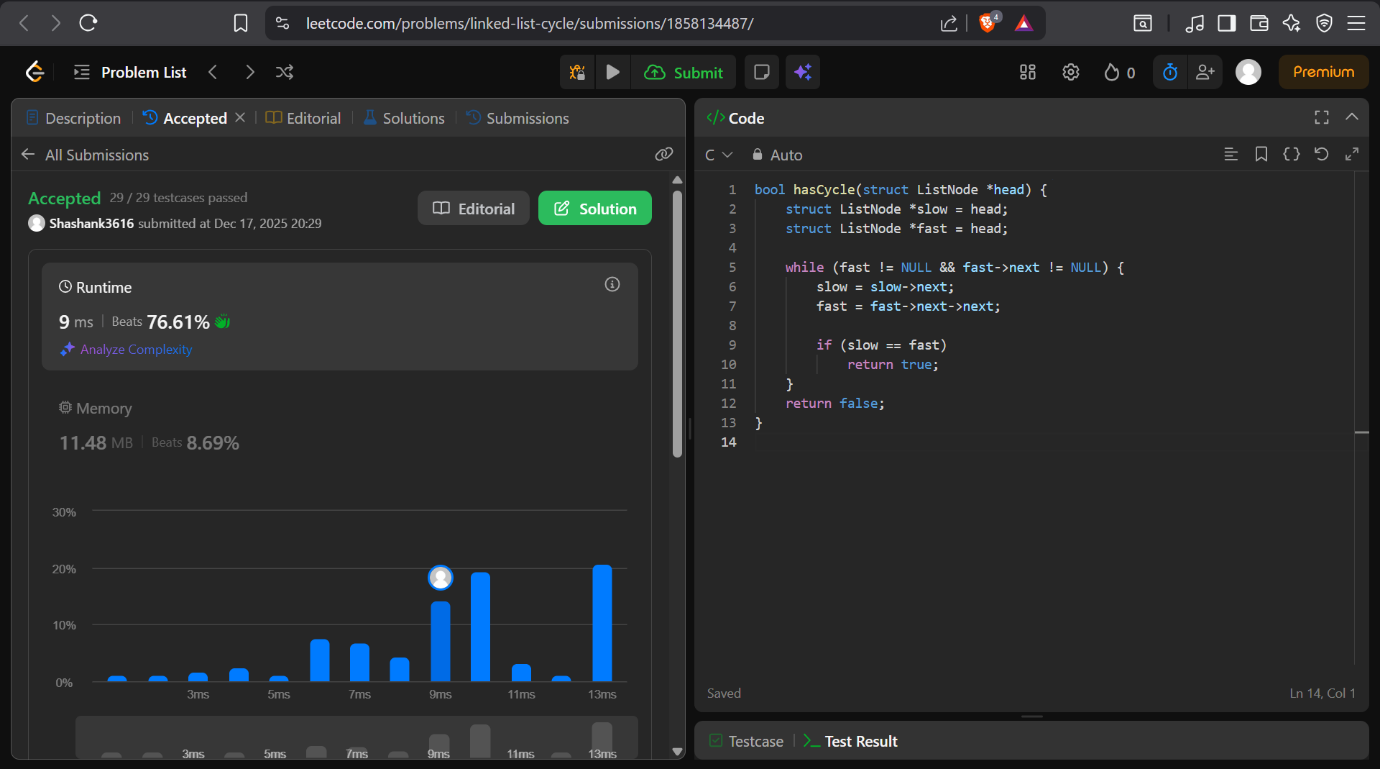
**if (slow == fast)**

**return true;**

**}**

**return false;**

**}**

****

**8] a]** **Write a program 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.**

**PROGRAMN:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*left;**

**struct Node \*right;**

**};**

**/\* Create a new node \*/**

**struct Node\* createNode(int item) {**

**struct Node \*newNode;**

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

**newNode->data = item;**

**newNode->left = newNode->right = NULL;**

**return newNode;**

**}**

**/\* Insert into BST \*/**

**struct Node\* insert(struct Node \*root, int item) {**

**if (root == NULL)**

**return createNode(item);**

**if (item < root->data)**

**root->left = insert(root->left, item);**

**else if (item > root->data)**

**root->right = insert(root->right, item);**

**return root;**

**}**

**/\* Inorder Traversal \*/**

**void inorder(struct Node \*root) {**

**if (root != NULL) {**

**inorder(root->left);**

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

**inorder(root->right);**

**}**

**}**

**/\* Preorder Traversal \*/**

**void preorder(struct Node \*root) {**

**if (root != NULL) {**

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

**preorder(root->left);**

**preorder(root->right);**

**}**

**}**

**/\* Postorder Traversal \*/**

**void postorder(struct Node \*root) {**

**if (root != NULL) {**

**postorder(root->left);**

**postorder(root->right);**

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

**}**

**}**

**int main() {**

**struct Node \*root = NULL;**

**int choice, item;**

**while (1) {**

**printf("\n===== BINARY SEARCH TREE MENU =====\n");**

**printf("1. Insert\n");**

**printf("2. Inorder Traversal\n");**

**printf("3. Preorder Traversal\n");**

**printf("4. Postorder Traversal\n");**

**printf("5. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

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

**scanf("%d", &item);**

**root = insert(root, item);**

**break;**

**case 2:**

**printf("Inorder Traversal: ");**

**inorder(root);**

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

**break;**

**case 3:**

**printf("Preorder Traversal: ");**

**preorder(root);**

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

**break;**

**case 4:**

**printf("Postorder Traversal: ");**

**postorder(root);**

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

**break;**

**case 5:**

**exit(0);**

**default:**

**printf("Invalid choice\n");**

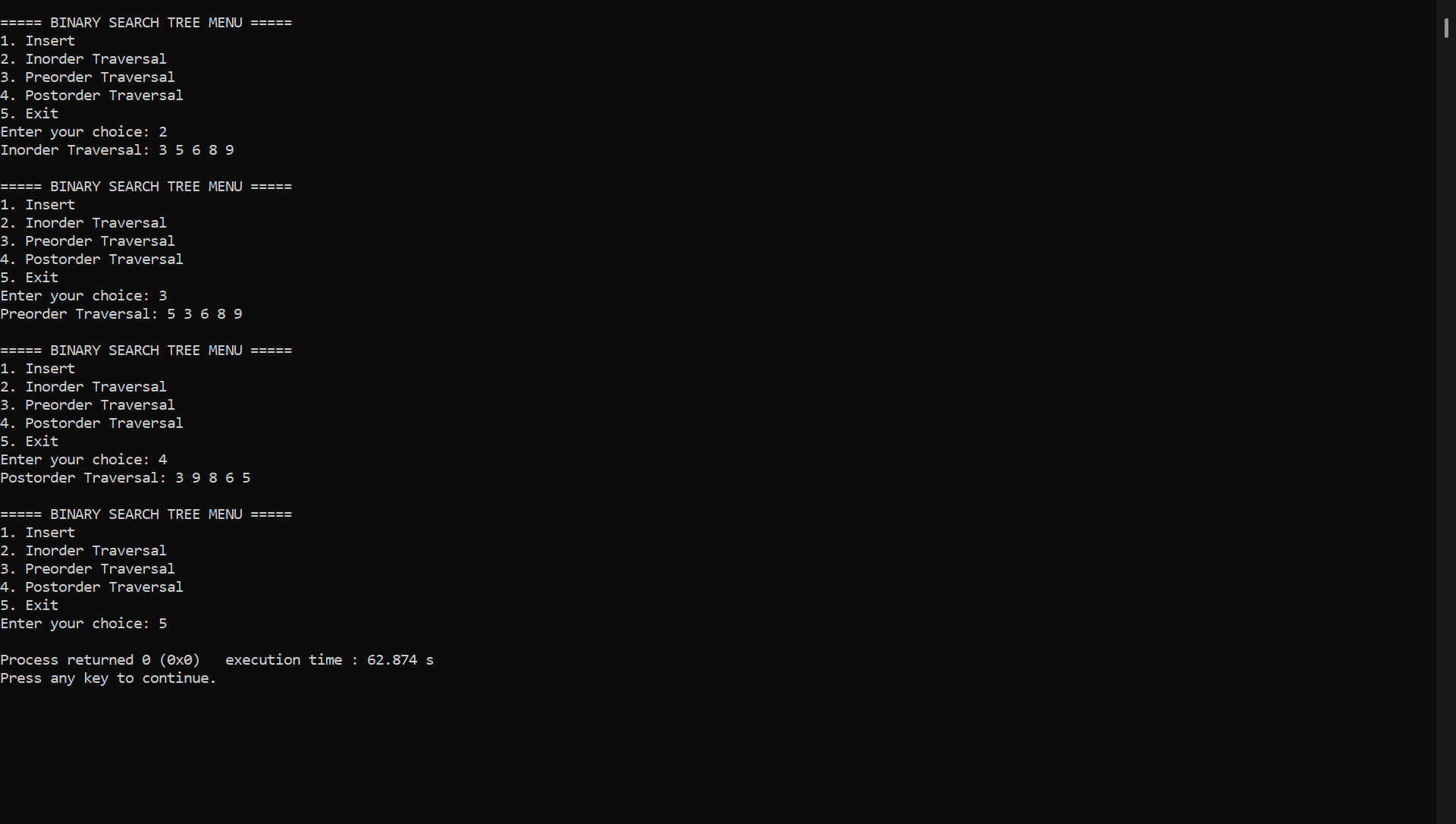
**}**

**}**

**return 0;**

**}**

****

****

**8] b] Program - Leetcode platform**

**You are given two binary trees root1 and root2.**

**Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.**

**Return *the merged tree*.**

**Note: The merging process must start from the root nodes of both trees.**

**FUNCTION:**

**struct TreeNode\* mergeTrees(struct TreeNode\* root1, struct TreeNode\* root2) {**

**if (root1 == NULL)**

**return root2;**

**if (root2 == NULL)**

**return root1;**

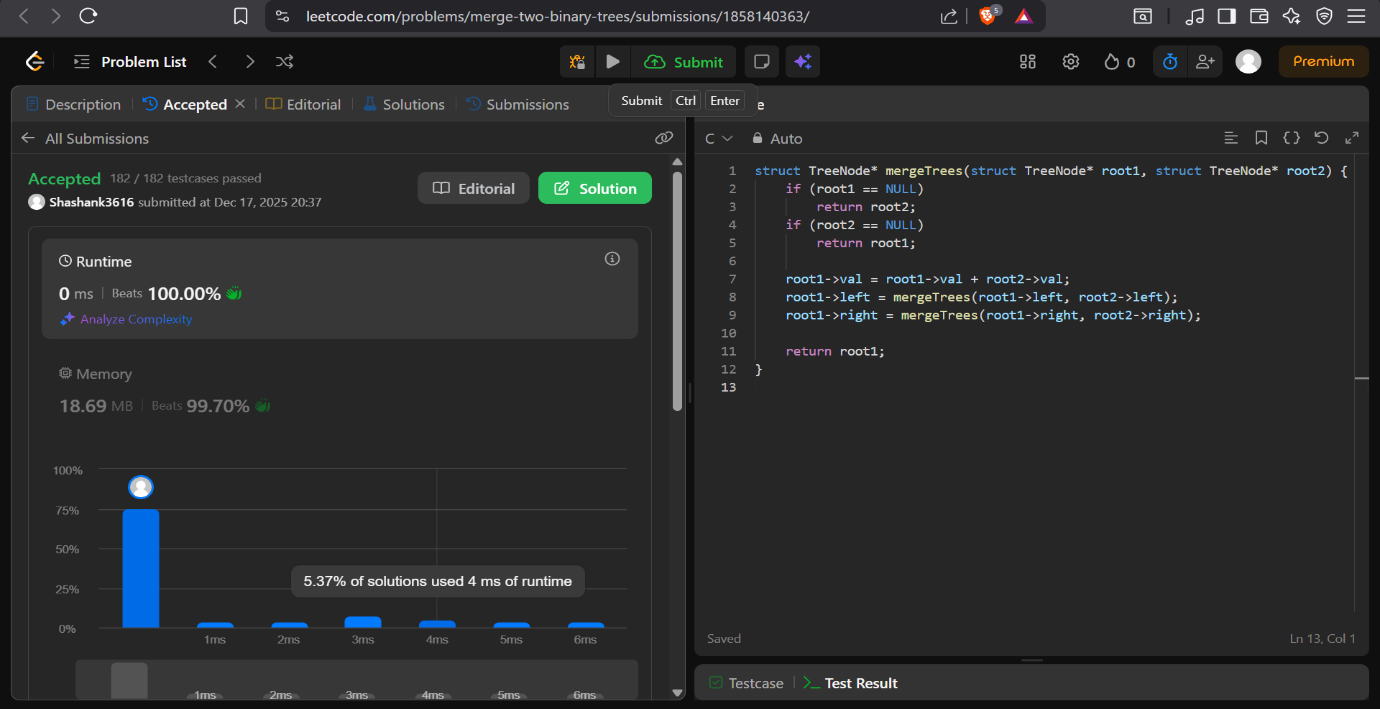
**root1->val = root1->val + root2->val;**

**root1->left = mergeTrees(root1->left, root2->left);**

**root1->right = mergeTrees(root1->right, root2->right);**

**return root1;**

**}**

****