VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

REEYA (1BM24CS238)

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 REEYA**(1BM24CS238)**, 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
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course outcomes:

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

1.Write a program to simulate the working of stack using an array with the following:

a) Push b) Pop c) Display

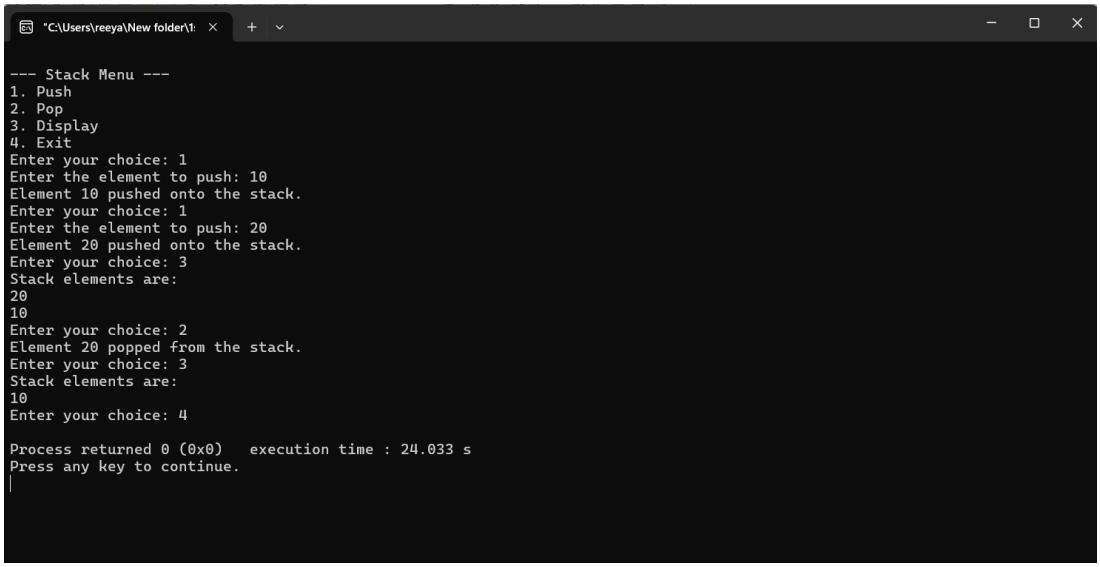
The program should print appropriate messages for stack overflow,stack underflow

```
#include <stdio.h>

#define MAX 5   // stack size


int stack[MAX];

int top = -1;
```

```
void push() {
    int item;
    if (top == MAX - 1) {
        printf("Stack Overflow! Cannot push element.\n");
    } else {
        printf("Enter the element to push: ");
        scanf("%d", &item);
        top++;
        stack[top] = item;
        printf("Element %d pushed into stack.\n", item);
    }
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow! Cannot pop element.\n");
    } else {
        printf("Popped element: %d\n", stack[top]);
        top--;
    }
}

void display() {
    int i;
    if (top == -1) {
```

```
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements are:\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: return 0;
        default: printf("Invalid choice! Try again.\n");
      }
   }
}
```



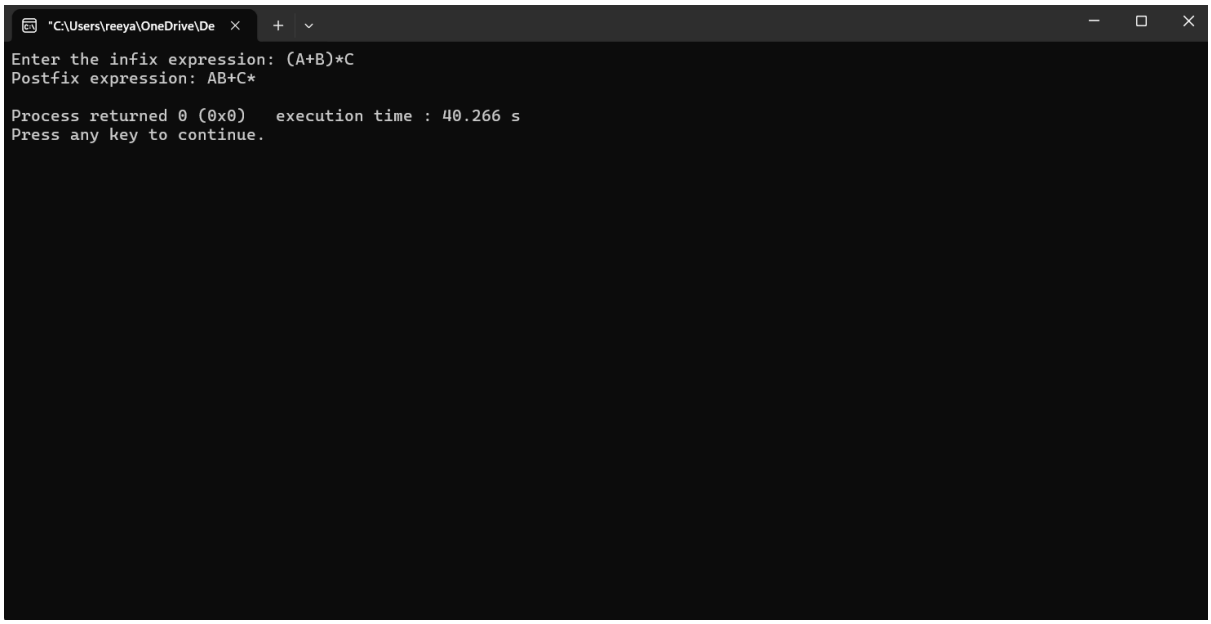
2. WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and /(divide)

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

#define MAX 100

char stack[MAX];
```

```
int top = -1;

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

char pop() {
   return stack[top--];
}

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

int main() {
   char infix[MAX], postfix[MAX];
   int i = 0, k = 0;
   char ch;

   printf("Enter the infix expression: ");
   scanf("%s", infix);
```

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

    if (isalnum(ch)) {
        postfix[k++] = ch;
    }


    else if (ch == '(') {
        push(ch);
    }


    else if (ch == ')') {
        while (stack[top] != '(') {
            postfix[k++] = pop();
        }
        pop();
    }


    else {
        while (top != -1 && precedence(stack[top]) >= precedence(ch)) {
            postfix[k++] = pop();
        }
        push(ch);
    }
}
```

```
    while (top != -1) {
        postfix[k++] = pop();
    }

    postfix[k] = '\0';

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

    return 0;
}
```



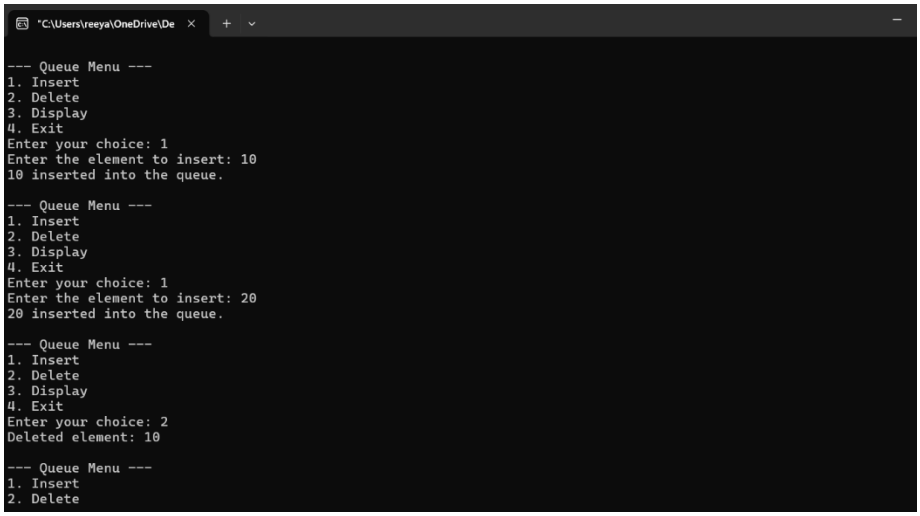
3.a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

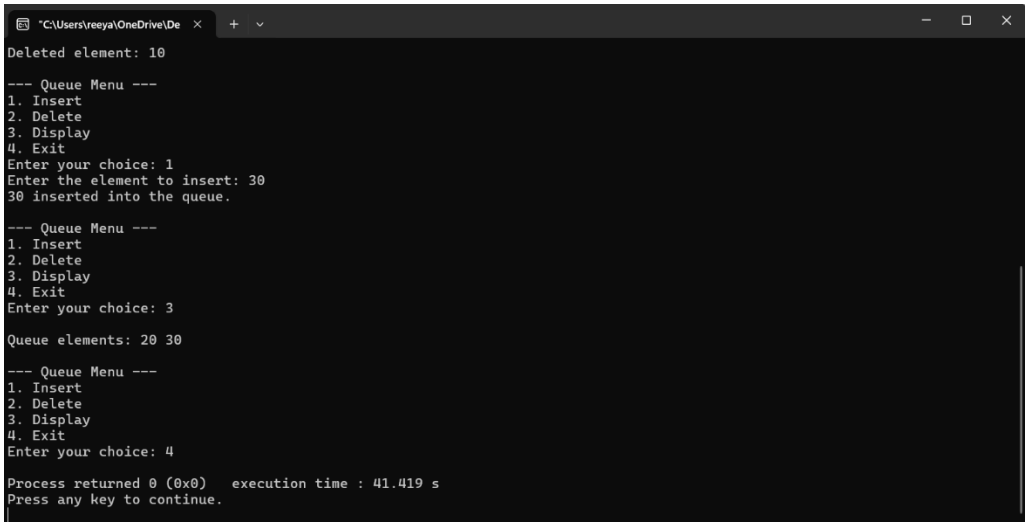
```
#include <stdio.h>

#define MAX 5   // maximum size of queue

int queue[MAX];

int front = -1;
```

```
int rear = -1;
void insert() {
   int item;
   if (rear == MAX - 1) {
      printf("\nQueue Overflow! Cannot insert.\n");
   } else {
      printf("Enter the element to insert: ");
      scanf("%d", &item);

      if (front == -1) {
         front = 0; // first element
      }
      rear++;
      queue[rear] = item;
      printf("%d inserted into the queue.\n", item);
   }
}
void delete() {
   if (front == -1 || front > rear) {
      printf("\nQueue Underflow! Queue is empty.\n");
   } else {
      printf("Deleted element: %d\n", queue[front]);
      front++;
      if (front > rear) {
         front = rear = -1;
      }
```

```
    }
}
void display() {
    if (front == -1) {
        printf("\nQueue is empty.\n");
    } else {
        printf("\nQueue elements: ");
        for (int 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:
            return 0;
        default:
            printf("Invalid choice! Try again.\n");
        }
    }
}
```





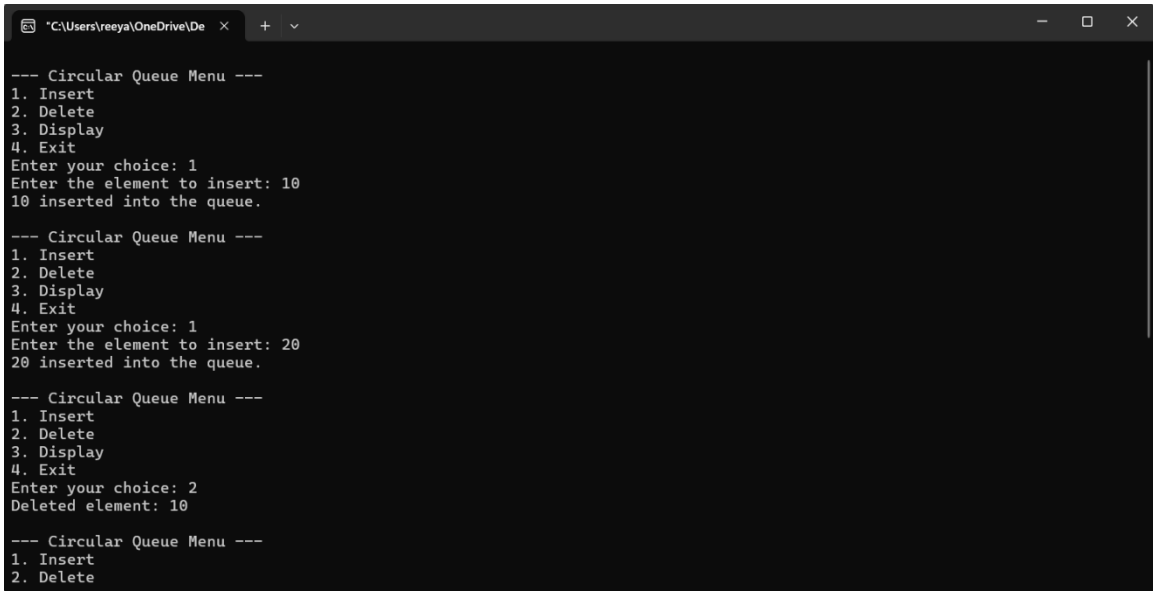
b.WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

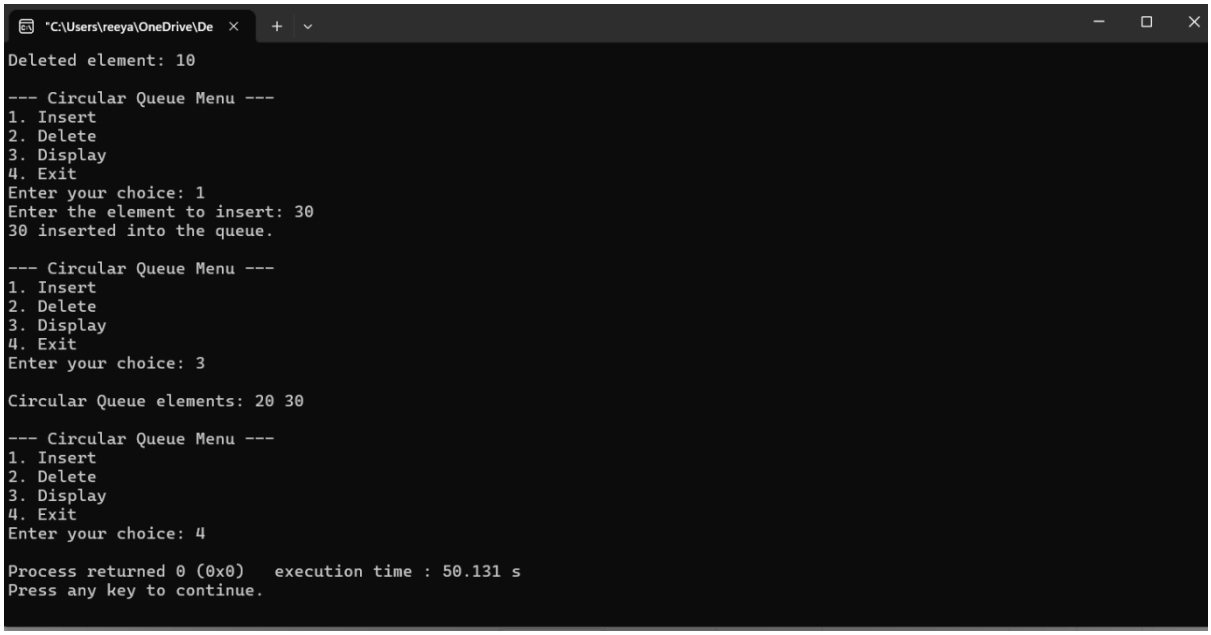
```
#include <stdio.h>

#define MAX 5


int queue[MAX];

int front = -1, rear = -1;

void insert() {
   int item;
   if ((rear + 1) % MAX == front) {
      printf("\nQueue Overflow! (Circular Queue is Full)\n");
      return;
   }


   printf("Enter the element to insert: ");
   scanf("%d", &item);


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

```
        front = rear = 0; // First element
    } else {
        rear = (rear + 1) % MAX;
    }

    queue[rear] = item;
    printf("%d inserted into the queue.\n", item);
}

// Function to delete (dequeue)
void delete() {
    if (front == -1) {
        printf("\nQueue Underflow! (Queue is Empty)\n");
        return;
    }

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

    if (front == rear) {
        // Only one element was present
        front = rear = -1;
    } else {
        front = (front + 1) % MAX;
    }
}
```

```
// Function to display queue
void display() {
    if (front == -1) {
        printf("\nQueue is Empty.\n");
        return;
    }

    printf("\nCircular Queue elements: ");
    int i = front;

    while (1) {
        printf("%d ", queue[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: return 0;
            default: printf("Invalid choice! Try again.\n");
        }
    }
}
```





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. Display the contents of the linked list.

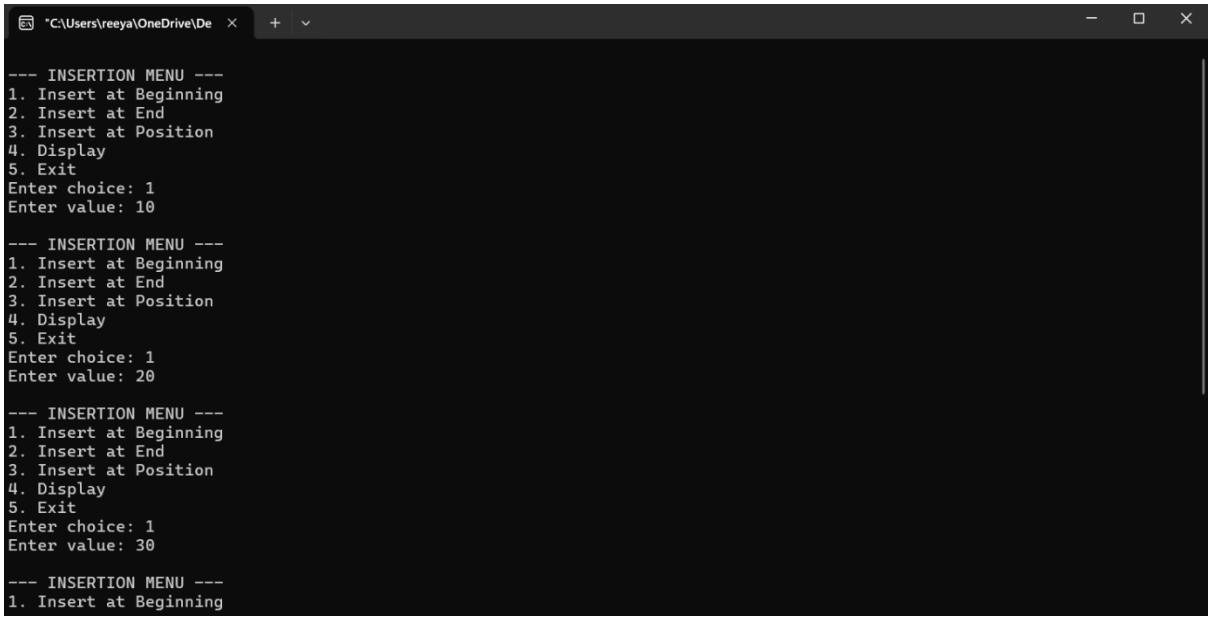
```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* next;

};


struct Node* head = NULL;

void insertAtBeginning(int value) {

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

   newNode->data = value;

   newNode->next = head;

   head = newNode;
```

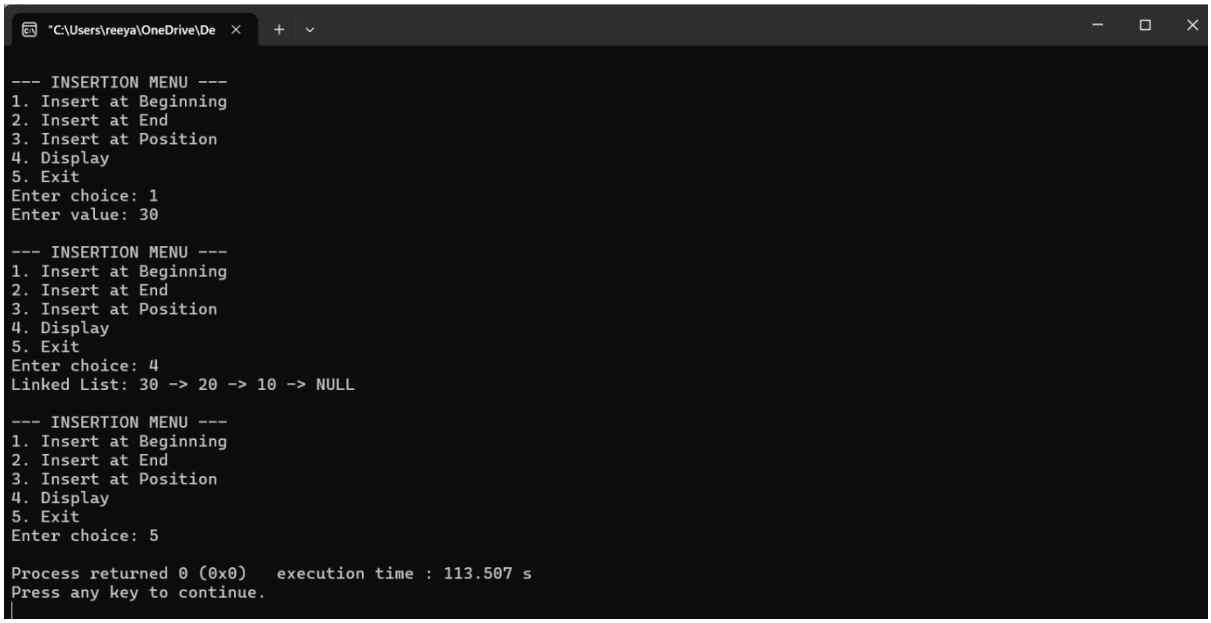
```
}
void insertAtEnd(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        return;
    }

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

    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        return;
    }
```

```
    struct Node* temp = head;
    for (int i = 1; i < pos - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Invalid position!\n");
        free(newNode);
        return;
    }

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

void display() {
    struct Node* temp = head;
    if (temp == 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, value, pos;

    while (1) {
        printf("\n--- INSERTION MENU ---\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

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

            case 2:
                printf("Enter value: ");
```

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

        case 3:
            printf("Enter value: ");
            scanf("%d", &value);
            printf("Enter position: ");
            scanf("%d", &pos);
            insertAtPosition(value, pos);
            break;

        case 4:
            display();
            break;

        case 5:
            return 0;

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





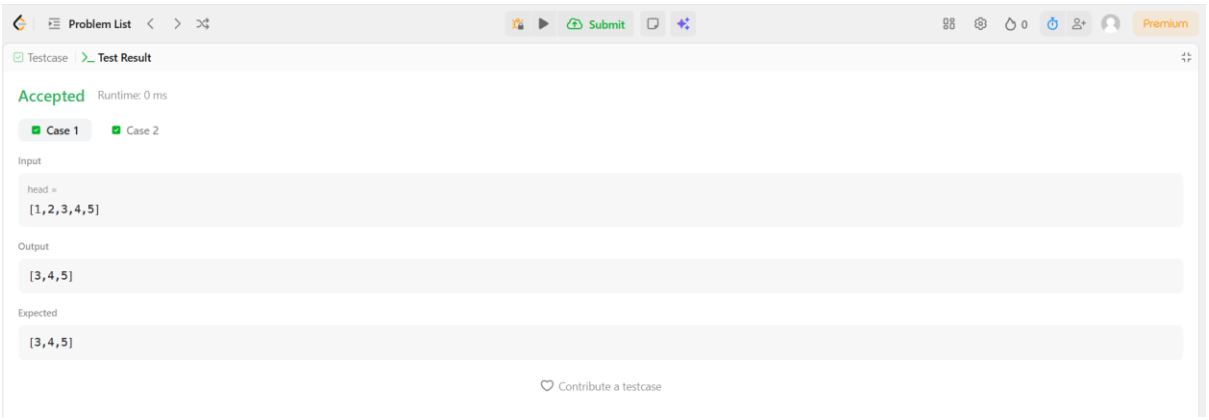
4b. Given the head of a singly linked list, return *the middle node of the linked list*.

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

Code

C Auto

```c

struct ListNode* middleNode(struct ListNode* head) {
    struct ListNode *slow = head;
    struct ListNode *fast = head;

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }

    return slow;
}

```



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.

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* next;
```

```
};

struct Node* head = NULL;

// Create linked list by inserting at end
void create(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
      return;
   }

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

   temp->next = newNode;
}
void deleteFirst() {
   if (head == NULL) {
      printf("List is empty!\n");
      return;
```

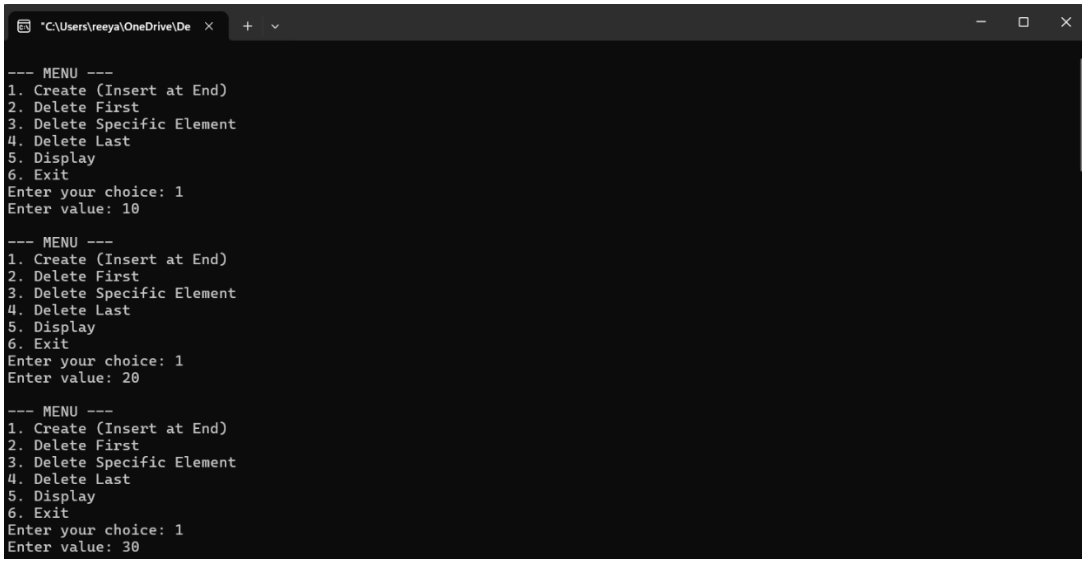
```
    }

    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("First node deleted.\n");
}
void deleteElement(int value) {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }

    // If element is at head
    if (head->data == value) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
        printf("Element %d deleted.\n", value);
        return;
    }

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

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

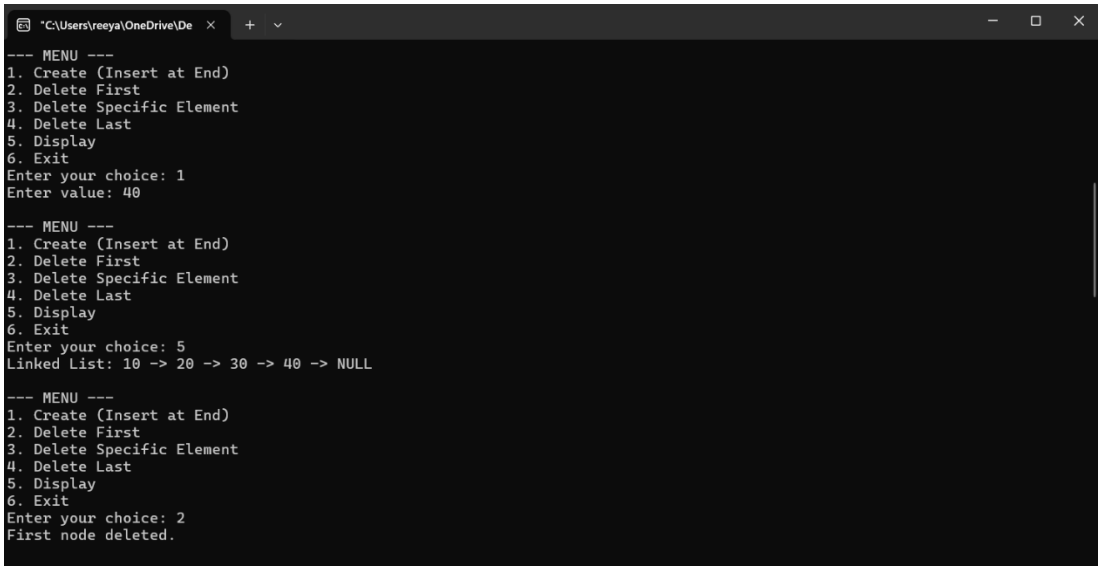
```
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Element %d not found!\n", value);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Element %d deleted.\n", value);
}

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

    if (head->next == NULL) {
        free(head);
        head = NULL;
        printf("Last node deleted.\n");
        return;
    }
```

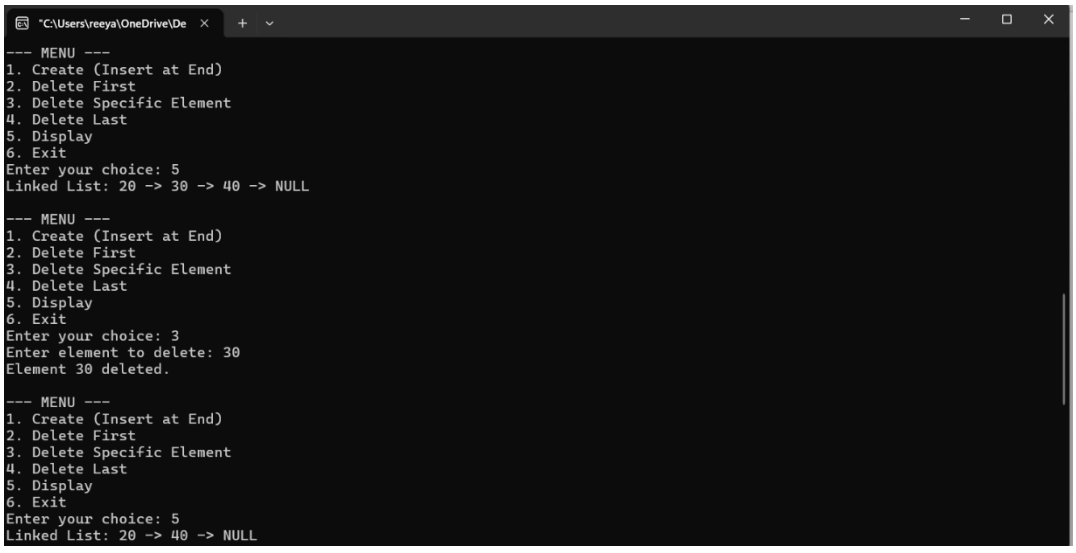
```
    struct Node* temp = head;
    struct Node* prev = NULL;

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

    prev->next = NULL;
    free(temp);
    printf("Last node deleted.\n");
}

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

    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
```

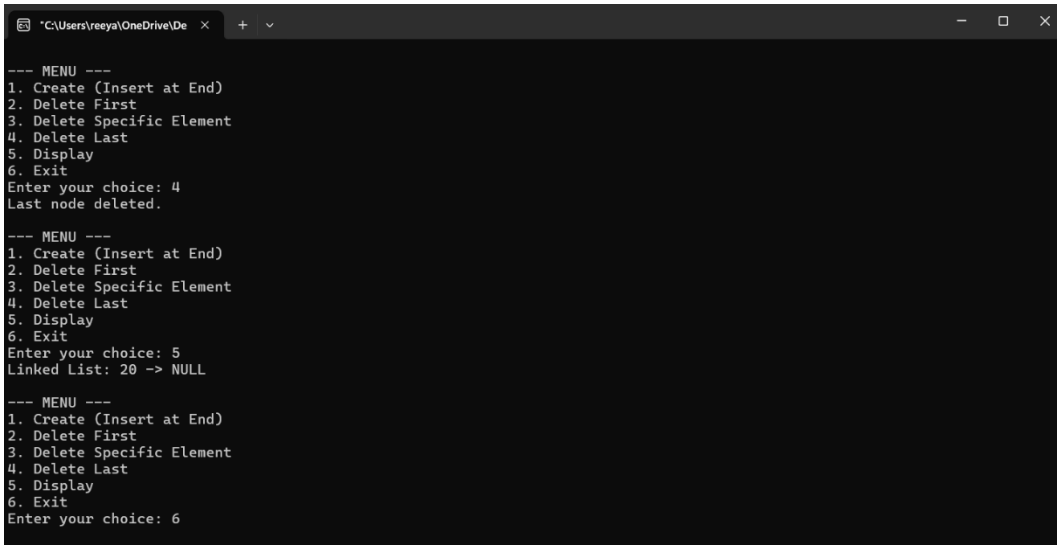
```
    printf("NULL\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\n--- MENU ---\n");
        printf("1. Create (Insert at End)\n");
        printf("2. Delete First\n");
        printf("3. Delete Specific 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:
                printf("Enter value: ");
                scanf("%d", &value);
                create(value);
                break;

            case 2:
                deleteFirst();
```

```
            break;

        case 3:
            printf("Enter element to delete: ");
            scanf("%d", &value);
            deleteElement(value);
            break;
      case 4:

          deleteLast();
            break;

        case 5:
            display();
            break;

        case 6:
            return 0;

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

   return 0;
}
```


"C:\Users\reeya\OneDrive\De
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 1
Enter value: 10
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 1
Enter value: 20
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 1
Enter value: 30


"C:\Users\reeya\OneDrive\De
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 1
Enter value: 40
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 5
Linked List: 10 -> 20 -> 30 -> 40 -> NULL
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 2
First node deleted.


"C:\Users\reeya\OneDrive\De
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 5
Linked List: 20 -> 30 -> 40 -> NULL
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 3
Enter element to delete: 30
Element 30 deleted.
--- MENU ---
1. Create (Insert at End)
2. Delete First
3. Delete Specific Element
4. Delete Last
5. Display
6. Exit
Enter your choice: 5
Linked List: 20 -> 40 -> NULL

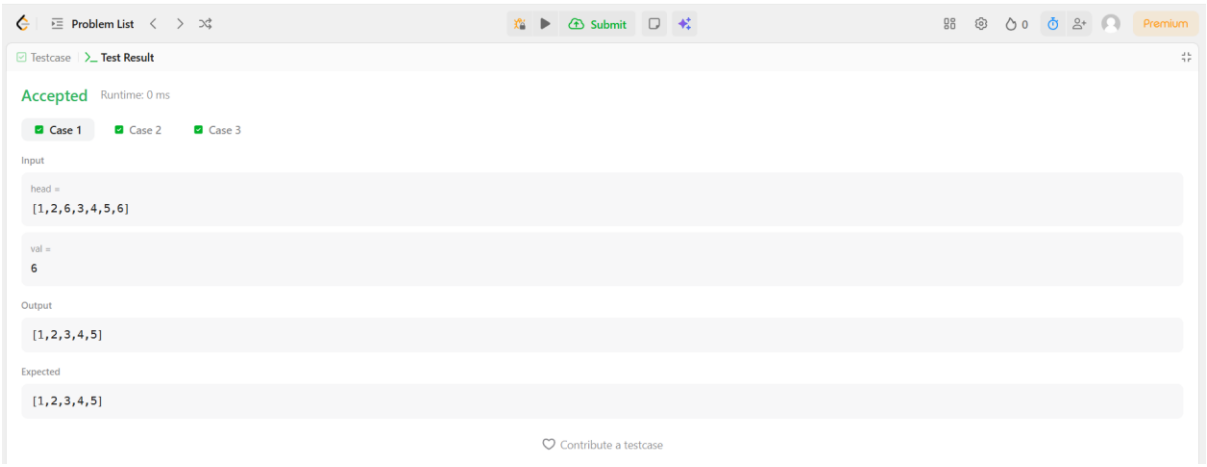


5bGiven the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*

Code

C Auto

```c
struct ListNode* removeElements(struct ListNode* head, int val) {
    // Remove nodes from beginning if needed
    while (head != NULL && head->val == val) {
        struct ListNode* temp = head;
        head = head->next;
        free(temp);
    }

    struct ListNode* curr = head;

    // Remove nodes from rest of the list
    while (curr != NULL && curr->next != NULL) {
        if (curr->next->val == val) {
            struct ListNode* temp = curr->next;
            curr->next = temp->next;
            free(temp);
        } else {
            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.

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

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

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


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

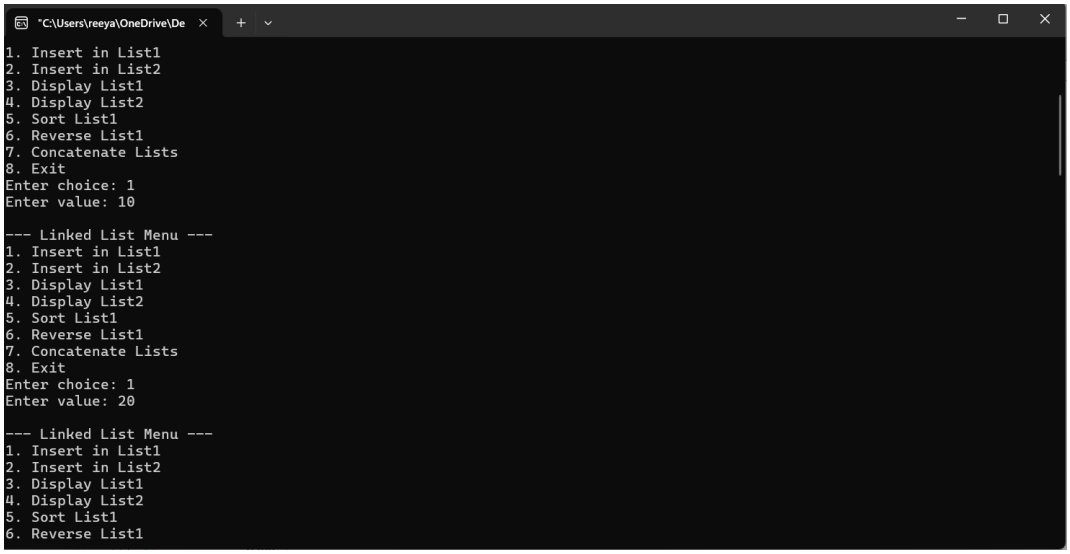
```
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node *temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}



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



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

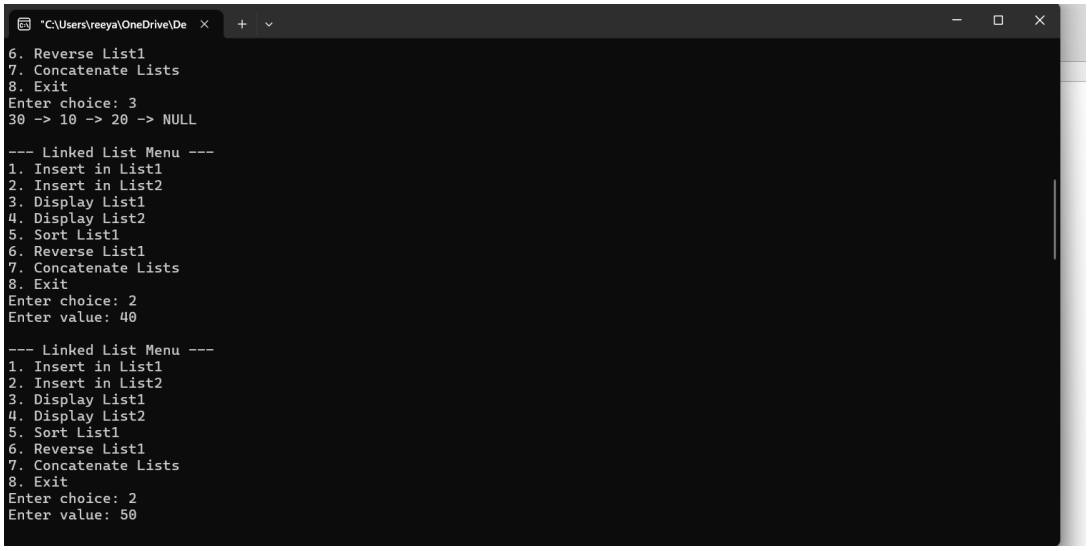
```
    struct Node *i, *j;
    int temp;
    if (head == NULL) return;

    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;
            }
        }
    }
}


struct Node* reverseList(struct Node *head) {
    struct Node *prev = NULL, *current = head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
```

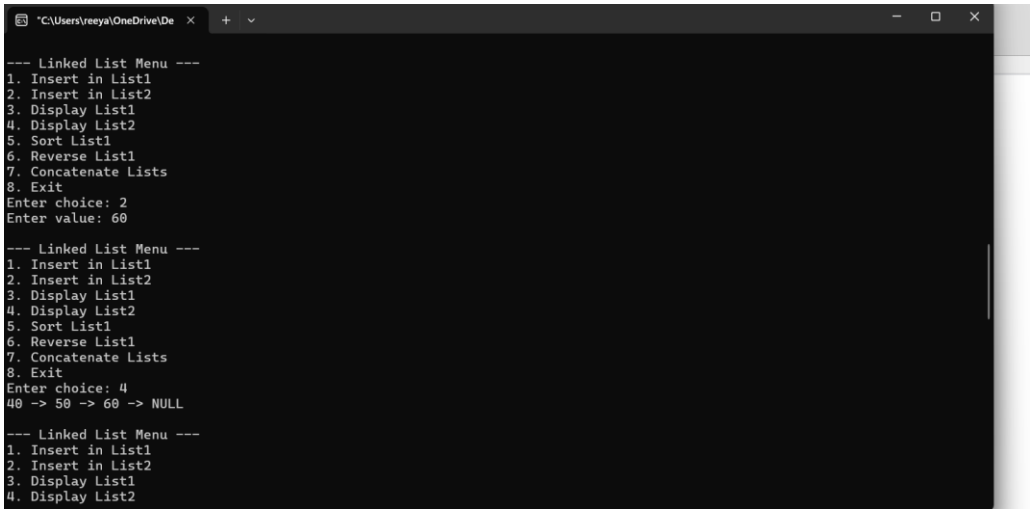
```
void concatenate() {
    if (head1 == NULL) {
        head1 = head2;
        return;
    }
    struct Node *temp = head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = head2;
}

int main() {
    int choice, val;

    while (1) {
        printf("\n--- Linked List Menu ---\n");
        printf("1. Insert in List1\n2. Insert in List2\n3. Display List1\n4. Display List2\n");
        printf("5. Sort List1\n6. Reverse List1\n7. Concatenate Lists\n8. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value: ");
```

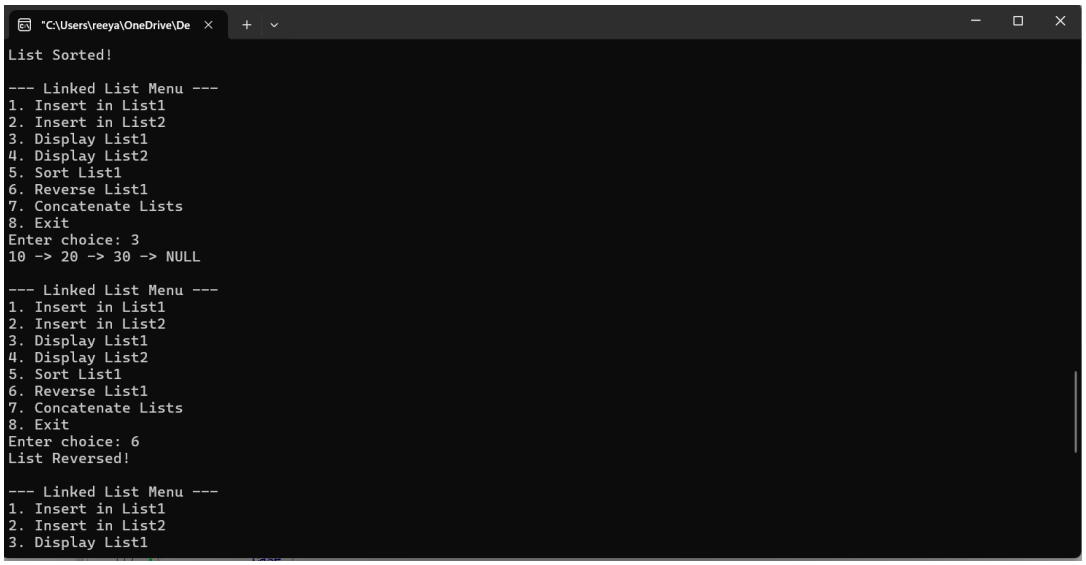
```
            scanf("%d", &val);
            insertEnd(&head1, val);
            break;
        case 2:
            printf("Enter value: ");
            scanf("%d", &val);
            insertEnd(&head2, val);
            break;
        case 3:
            display(head1);
            break;
        case 4:
            display(head2);
            break;
        case 5:
            sortList(head1);
            printf("List Sorted!\n");
            break;
        case 6:
            head1 = reverseList(head1);
            printf("List Reversed!\n");
            break;
        case 7:
            concatenate();
            printf("Concatenated! Updated List1: ");
            display(head1);
```

```
            break;
        case 8:
            exit(0);
        default:
            printf("Invalid Choice!\n");
      }
   }
}
```









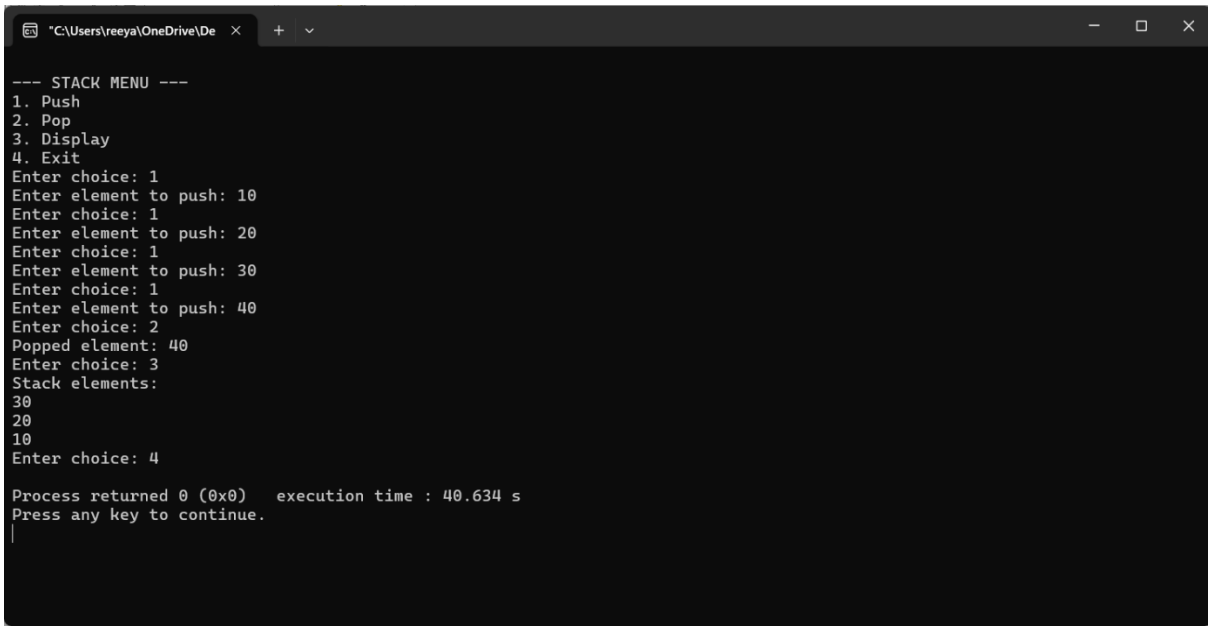
b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

#include <stdio.h>

```
#include <stdlib.h>

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

struct node *top = NULL;

/* Push */
void push() {
   int x;
   struct node *p = (struct node *)malloc(sizeof(struct node));
   printf("Enter element to push: ");
   scanf("%d", &x);
   p->data = x;
   p->next = top;
   top = p;
}

/* Pop */
void pop() {
   if (top == NULL) {
      printf("Stack is empty\n");
      return;
   }
```

```
    struct node *temp = top;
    printf("Popped element: %d\n", temp->data);
    top = temp->next;
    free(temp);
}

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

int main() {
    int choice;
    printf("\n--- STACK MENU ---\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");

    while (1) {
```

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

        switch (choice) {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: displayStack(); break;
            case 4: exit(0);
            default: printf("Invalid choice\n");
        }
    }
}
```



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

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

```
#include <stdlib.h>

// Structure definition
struct node {
   int data;
   struct node *prev;
   struct node *next;
};

struct node *head = NULL;

// Create doubly linked list
void create(int n) {
   struct node *newnode, *temp;
   int data;

   for (int i = 0; i < n; i++) {
      newnode = (struct node *)malloc(sizeof(struct node));
      printf("Enter data: ");
      scanf("%d", &data);

      newnode->data = data;
      newnode->prev = NULL;
      newnode->next = NULL;
```

```
        if (head == NULL) {
            head = newnode;
            temp = head;
        } else {
            temp->next = newnode;
            newnode->prev = temp;
            temp = newnode;
        }
    }
}

// Insert to the left of a given value
void insert_left(int value, int newdata) {
    struct node *temp = head;
    struct node *newnode;

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

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

    if (temp == NULL) {
```

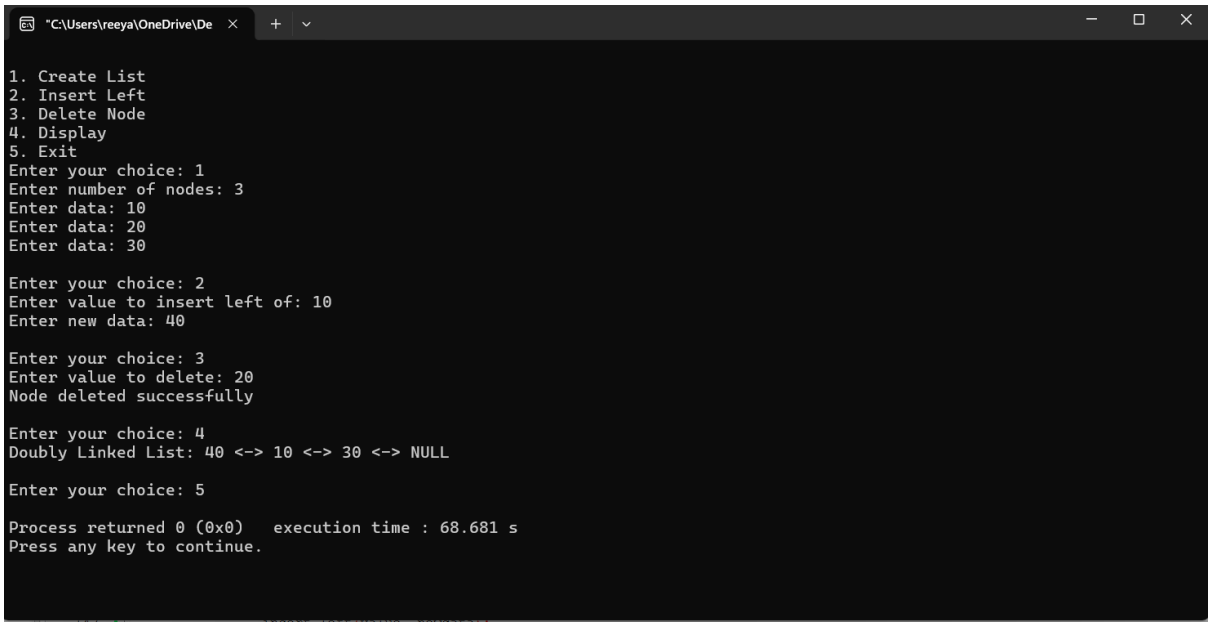
```
        printf("Value not found\n");
        return;
    }

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

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

    if (temp->prev != NULL)
        temp->prev->next = newnode;
    else
        head = newnode;

    temp->prev = newnode;
}

// Delete node with specific value
void delete_value(int value) {
    struct node *temp = head;

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

```
    }

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

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

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

    if (temp->next != NULL)
        temp->next->prev = temp->prev;

    free(temp);
    printf("Node deleted successfully\n");
}

// Display the list
void display() {
    struct node *temp = head;
```

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

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

// Main function
int main() {
    int n, choice, value, newdata;
     printf("\n1. Create List");
     printf("\n2. Insert Left");
     printf("\n3. Delete Node");
     printf("\n4. Display");
     printf("\n5. Exit");

    while (1) {
```

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

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

   case 2:
      printf("Enter value to insert left of: ");
      scanf("%d", &value);
      printf("Enter new data: ");
      scanf("%d", &newdata);
      insert_left(value, newdata);
      break;

   case 3:
      printf("Enter value to delete: ");
      scanf("%d", &value);
      delete_value(value);
      break;

   case 4:
```

```
            display();
            break;

        case 5:
            exit(0);

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



7b. 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

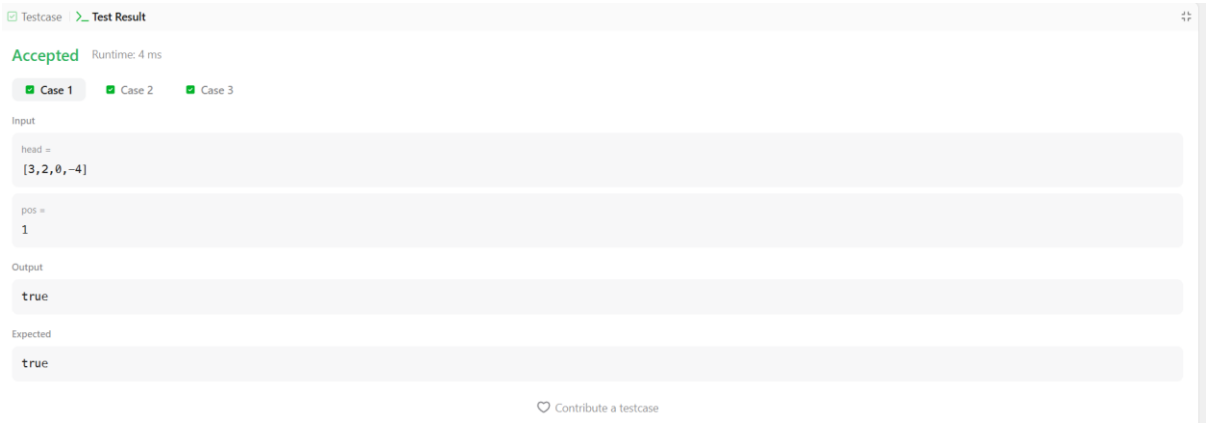
used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true *if there is a cycle in the linked list*. Otherwise, return false.

Code

C Auto

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */

bool hasCycle(struct ListNode *head) {
    struct ListNode *slow = head;
    struct ListNode *fast = head;

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;          // move 1 step
        fast = fast->next->next;    // move 2 steps

        if (slow == fast) {
            return true;            // cycle found
        }
    }

    return false;   // no cycle
}
```



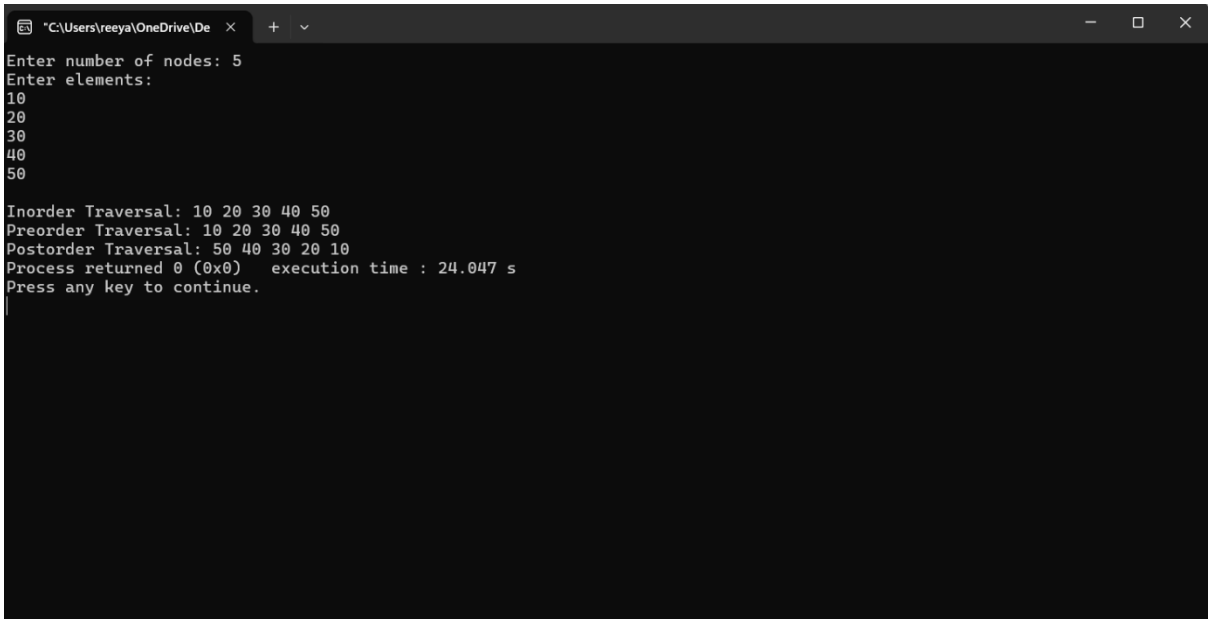
8.Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree.

```
#include <stdio.h>

#include <stdlib.h>


// Structure definition

struct node {

   int data;

   struct node *left;

   struct node *right;

};


// Create a new node

struct node* createNode(int data) {

   struct node *newnode;

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

   newnode->data = data;

   newnode->left = NULL;

   newnode->right = NULL;

   return newnode;

}


// Insert a node into BST

struct node* insert(struct node *root, int data) {
```

```
    if (root == NULL)
        return createNode(data);

    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);

    return root;
}

// 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);
    }
}

// Main function
int main() {
    struct node *root = NULL;
    int n, data;

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

    printf("Enter elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        root = insert(root, data);
```

```
    }

    printf("\nInorder Traversal: ");
    inorder(root);

    printf("\nPreorder Traversal: ");
    preorder(root);

    printf("\nPostorder Traversal: ");
    postorder(root);

    return 0;
}
```



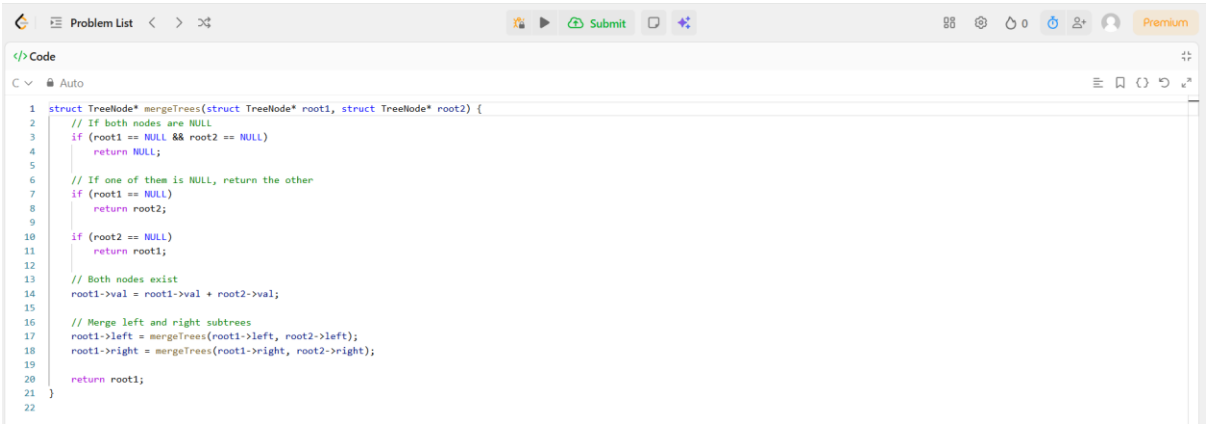
8b. You are given two binary trees root1 and root2.

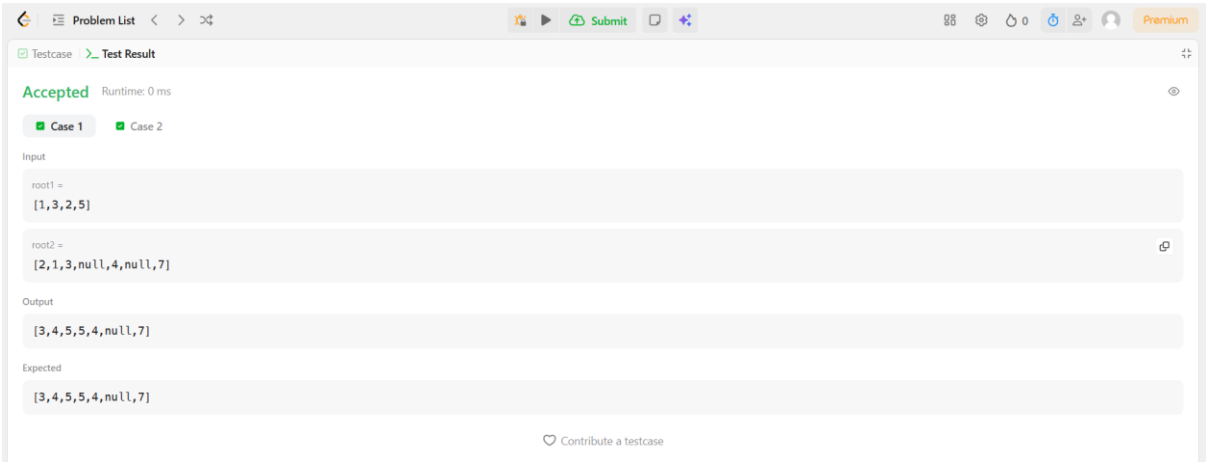
Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes

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.



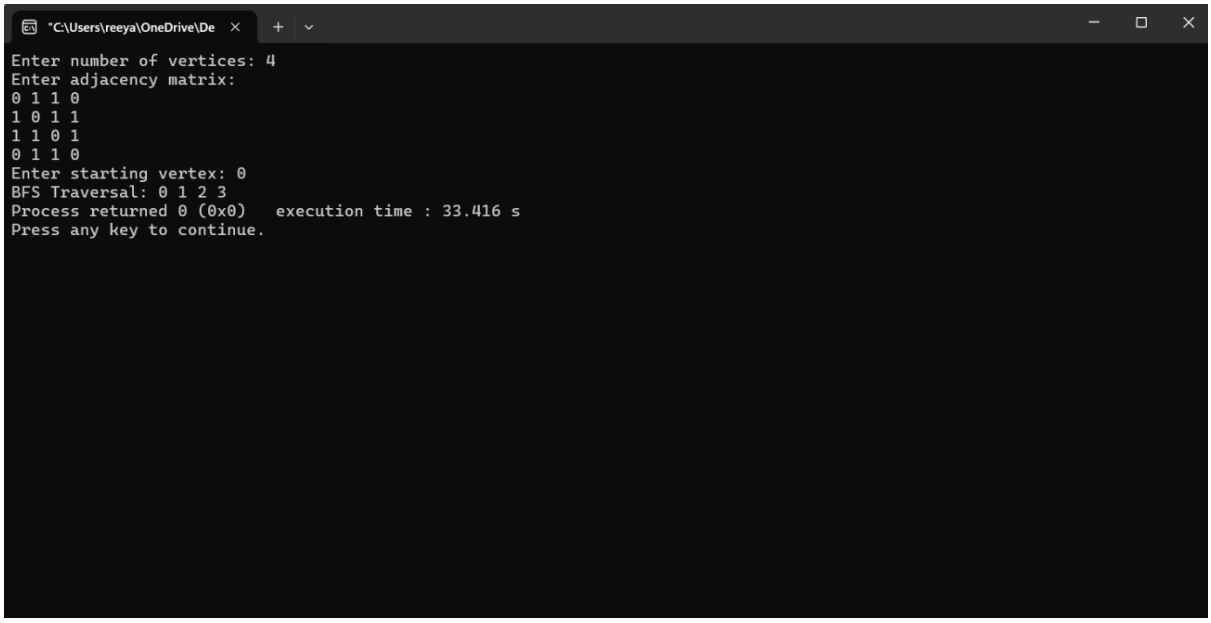


9. a) Write a program to traverse a graph using BFS method.

#include <stdio.h>

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

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

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

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

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

int main() {
   int start;
```

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

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

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

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

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

    return 0;
}
```

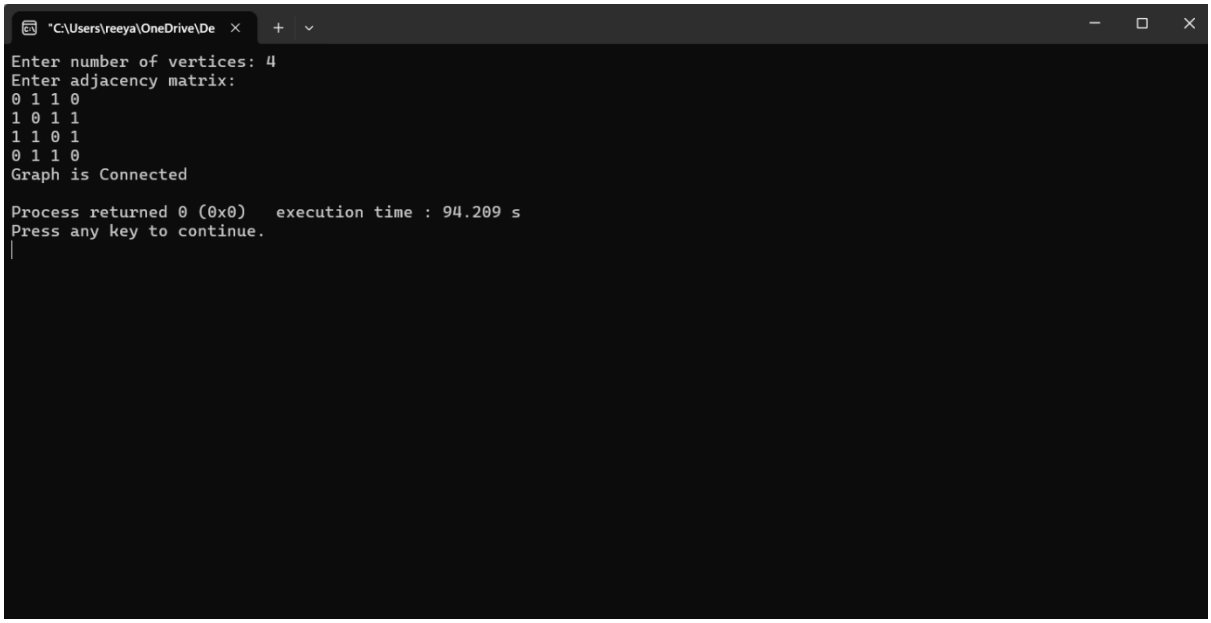


b) Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>


#define MAX 10


int adj[MAX][MAX];

int visited[MAX];

int n;


void DFS(int v) {

   visited[v] = 1;


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

      if (adj[v][i] == 1 && !visited[i]) {

         DFS(i);
```

```
        }
    }
}

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

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

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

    DFS(0);

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            printf("Graph is NOT Connected\n");
            return 0;
```

```
        }
    }

    printf("Graph is Connected\n");
    return 0;
}
```



10. Given a File of N employee records with a set K of Keys(4 digit) which uniquely determine the records in file F. 10 5 10 Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 20
```

```
int hashTable[MAX];
int m;

// Initialize hash table
void init() {
   for (int i = 0; i < m; i++)
      hashTable[i] = -1;
}

// Hash function
int hashFunction(int key) {
   return key % m;
}

// Insert using Linear Probing
void insert(int key) {
   int index = hashFunction(key);

   if (hashTable[index] == -1) {
      hashTable[index] = key;
   } else {
      // Collision occurred
      int i = (index + 1) % m;
      while (i != index) {
         if (hashTable[i] == -1) {
```

```
            hashTable[i] = key;
            return;
        }
        i = (i + 1) % m;
    }
    printf("Hash table is full. Cannot insert key %d\n", key);
  }
}

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

// Main function
int main() {
  int n, key;

  printf("Enter size of hash table (m): ");
```

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

    init();

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

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

    display();
    return 0;
}
```