VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by
JAYANTH M M
(1BM25CS487-T)

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 **JAYANTH M M (1BM25CS487-T),** 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.

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

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

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Stack Operations: Push, Pop and Display Leetcode: 1, 26, 35	4
2	Infix to postfix	10
3	Queue operations (a) Linear Queue (b) Circular Queue	13
4	Singly Linked List Insertion Methods	18
5	Singly Linked List Deletion Methods	21
6	SLL with(a) Sorting, Reversing, Concatenation (b)Stack and Queue Operations using singly linked list Leetcode (a) Linked List Cycle 141 (b) Remove nth node from end of the list	25
7	(a) Doubly link list with primitive operations (b) Leetcode palindrome linked list 234	32
8	(a) Binary search tree i) Inorder ii) postorder iii) preorder (b) Leetcode Find if the path exists in graph 1971	37
9	(a)Traverse a graph using BFS method. (b) to Check Whether Graph is Connected Using DFS method	42
10	Linear probing	46

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>
#define SIZE 5
int stack[SIZE];

int top = -1;
// Function to push an element into the stack
void push(int value) {
    if (top == SIZE - 1) {
        printf("Stack Overflow! Cannot push %d\n", value);
    } else {
        top++;
        stack[top] = value;
        printf("%d pushed into stack\n", value);
    }
}

// Function to pop an element from the stack
void pop() {
    if (top == -1) {
        printf("Stack Underflow! No element to pop\n");
    } else {
        printf("%d popped from stack\n", stack[top]);
        top--;
    }
}

// Enhanced function to display stack elements
void display() {
    if (top == -1) {
        printf("Stack is empty!\n");
    } else {
        printf("\nCurrent Stack (Top to Bottom):\n");
        for (int i = top; i >= 0; i--) {
            if (i == top) {
                printf("| %d |  <- Top\n", stack[i]);
```

```
            } else {
                printf("| %d |\n", stack[i]);
            }
        }
        printf(" ----- \n");
    }
}

// Main function to run the menu-driven stack program
int main() {
    int choice, value;
    while (1) {
        printf("\n---- Stack Menu ----\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display Stack\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }

    return 0;
}
```

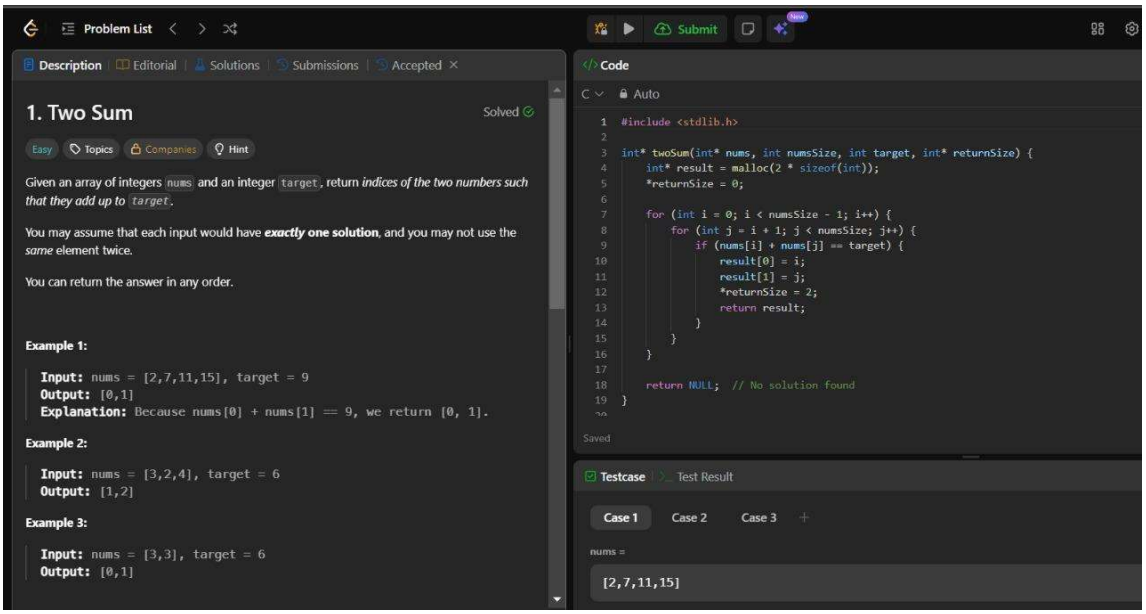
Output:

```
---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 2
Stack Underflow! No element to pop

---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 1
Enter value to push: 10
10 pushed into stack

---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 1
Enter value to push: 20
20 pushed into stack

---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 1
Enter value to push: 30
30 pushed into stack

---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 3

Current Stack (Top to Bottom):
| 30 |  <- Top
| 20 |
| 10 |
 -----

---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 2
30 popped from stack

---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 2
20 popped from stack

---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 3

Current Stack (Top to Bottom):
| 10 |  <- Top
 -----

---- Stack Menu ----
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 4
Exiting program.
```

LEETCODE-1

TWO SUM

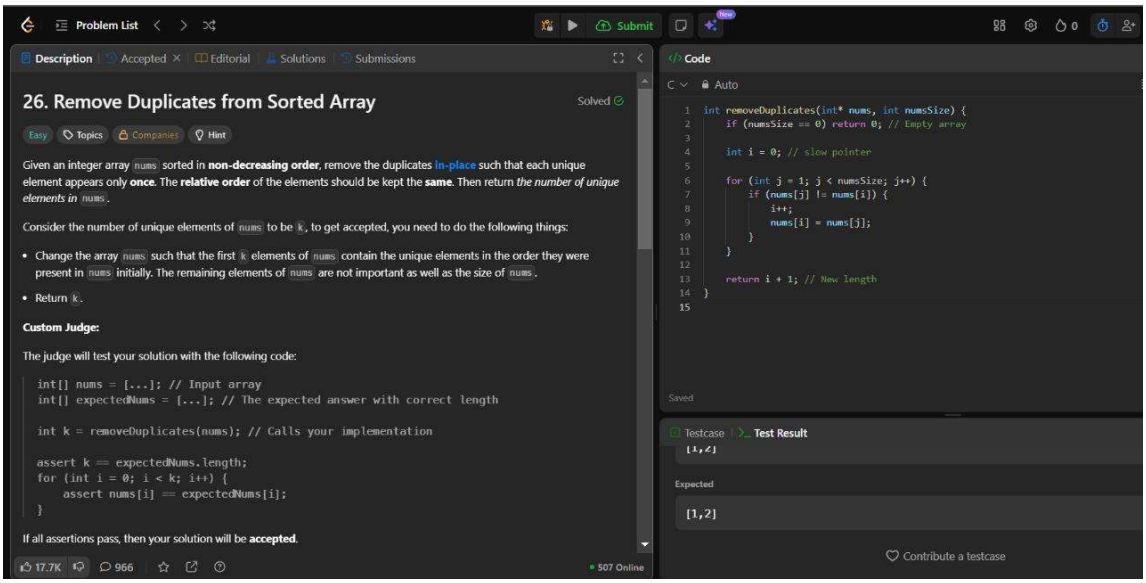
```
#include <stdlib.h>

int* twoSum(int* nums, int numsSize, int target, int* returnSize) {
    int* result = malloc(2 * sizeof(int));
    *returnSize = 0;

    for (int i = 0; i < numsSize - 1; i++) {
        for (int j = i + 1; j < numsSize; j++) {
            if (nums[i] + nums[j] == target) {
                result[0] = i;
                result[1] = j;
                *returnSize = 2;
                return result;
            }
        }
    }

    return NULL;  // No solution found
}
```



LEETCODE-26

Remove Duplicates from Sorted Array

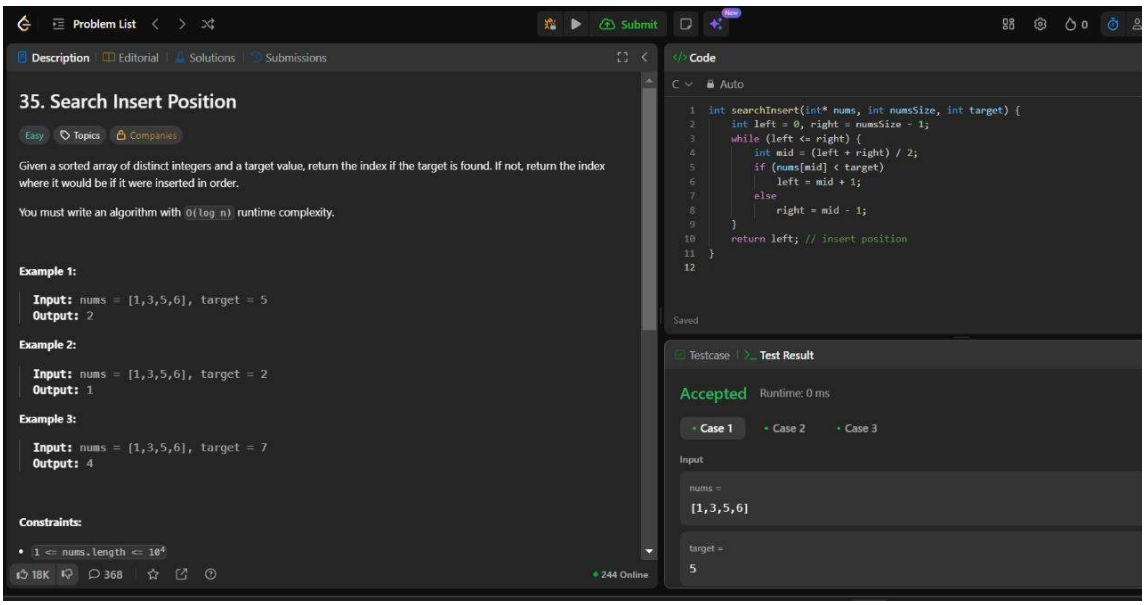
```
int removeDuplicates(int* nums, int numsSize) {
    if (numsSize == 0) return 0; // Empty array

    int i = 0; // slow pointer

    for (int j = 1; j < numsSize; j++) {
        if (nums[j] != nums[i]) {
            i++;
            nums[i] = nums[j];
        }
    }

    return i + 1; // New length
}
```



LEETCODE-35

Search Insert Position

```
int searchInsert(int* nums, int numsSize, int target) {
    int left = 0, right = numsSize - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (nums[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return left; // insert position
}
```



Lab Program 2

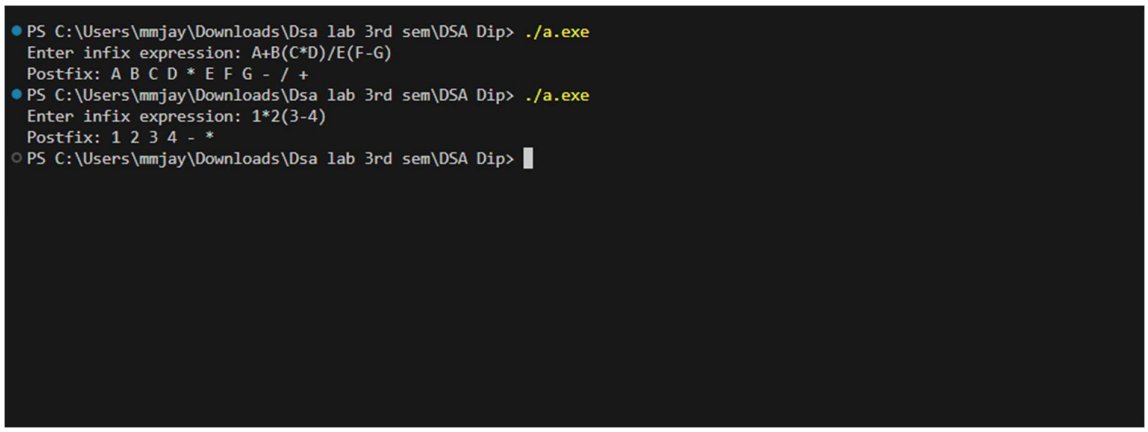
(a) WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>
#include <ctype.h>   // for isalnum() and isdigit()

#define MAX 100      // maximum size of the stack and input

// ---- Stack for operators ----
char stack[MAX];
int top = -1;        // top of stack ( -1 means empty )

// push one character onto the stack
void push(char c) {
   stack[++top] = c;
}

// pop one character from the stack
char pop() {
   return stack[top--];
}

// look at the top character without removing it
char peek() {
   return stack[top];
}

// returns precedence level of operators
int precedence(char op) {
   if(op == '^') return 3;              // highest
   if(op == '*' || op == '/') return 2;  // medium
   if(op == '+' || op == '-') return 1;  // lowest
   return 0;
}

int main() {
   char infix[MAX];

   // ask the user to enter an expression
   printf("Enter infix expression: ");
```

```
scanf("%[^\n]", infix);  // read entire line until ENTER

// push an opening '(' so we have a marker in the stack
push('(');

// find length of infix string
int len = 0;
while(infix[len] != '\0') len++;

// add a closing ')' at the end (so we pop everything finally)
infix[len] = ')';
infix[len+1] = '\0';

printf("Postfix: ");

// go through each character in the infix expression
for(int i = 0; infix[i] != '\0'; ) {

   // ignore spaces if user typed them
   if(infix[i] == ' ') {
      i++;
      continue;
   }

   // ---- If current is a letter or digit (part of number/variable) ----
   if(isalnum(infix[i]) || infix[i]=='.') {
      // print the entire number or variable (can be many chars)
      while(isalnum(infix[i]) || infix[i]=='.') {
         printf("%c", infix[i]);
         i++;
      }
      printf(" "); // add a space after the operand
   }

   // ---- If '(' just push to stack ----
   else if(infix[i] == '(') {
      push('(');
      i++;
   }

   // ---- If ')' pop until we find '(' ----
   else if(infix[i] == ')') {
```

```
            while(peek() != '(') {          // pop operators
                printf("%c ", pop());
            }
            pop(); // remove the '('
            i++;
        }

        // ---- If operator (+ - * / ^) ----
        else {
            char op = infix[i];
            // while top of stack has higher or equal precedence
            while(precedence(peek()) >= precedence(op)) {
                printf("%c ", pop());
            }
            // push the current operator
            push(op);
            i++;
        }
    }

    // because we put an extra ')' we don't need another loop
    printf("\n");
    return 0;
    }
```

Output:



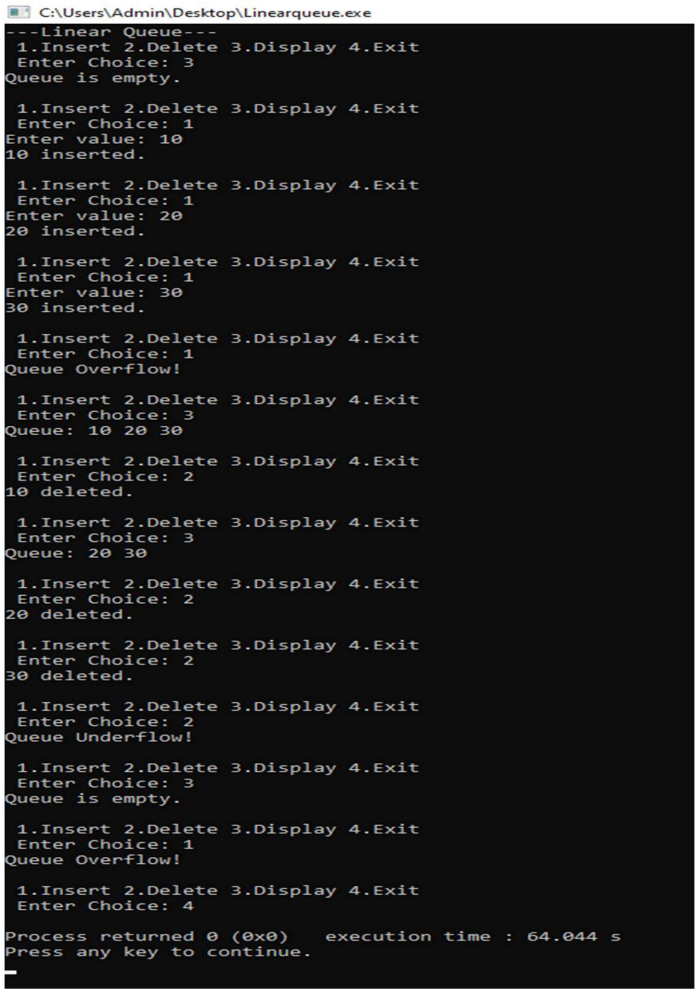
Lab Program 3

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 SIZE 3
int q[SIZE], front=-1, rear=-1;

void insert() {
   if (rear == SIZE-1) {
      printf("Queue Overflow!\n");
      return;
   }
   int x;
   printf("Enter value: ");
   scanf("%d",&x);
   if (front == -1) front = 0;
   q[++rear] = x;
   printf("%d inserted.\n", x);
}

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

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

int main() {
   int ch;
```

```
 printf("---Linear Queue---");
while(1) {
   printf("\n 1.Insert 2.Delete 3.Display 4.Exit\n Enter Choice: ");
   scanf("%d",&ch);
   switch(ch) {
      case 1: insert(); break;
      case 2: delete(); break;
      case 3: display(); break;
      case 4: return 0;
      default: printf("Invalid choice!\n");
   }
}}
```

Output:

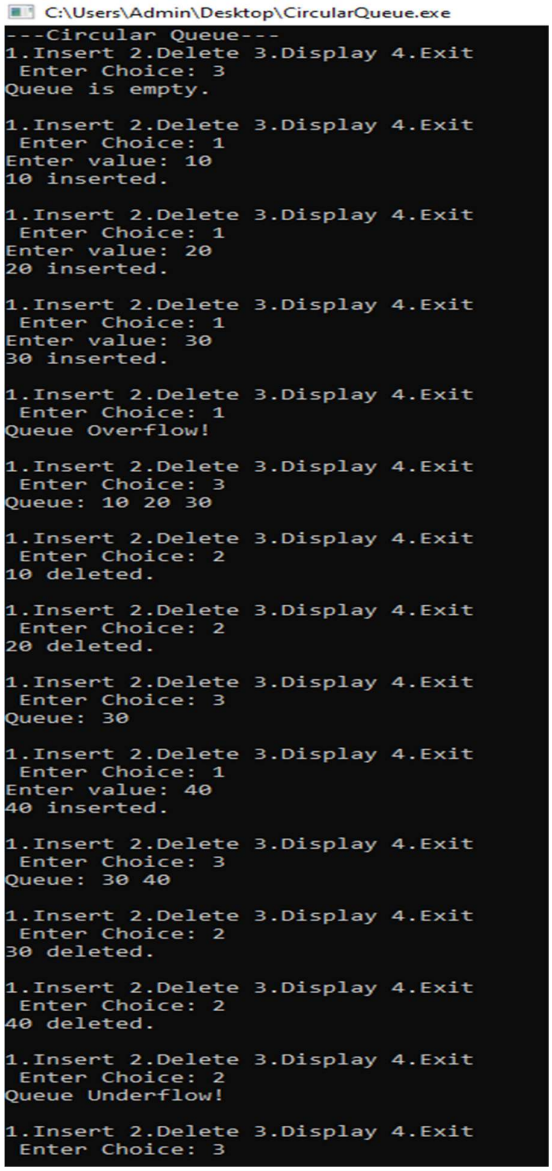


b) WAP to simulate the working of Circular Queue using an array with the following operations: Insert, Delete and Display, also should print appropriate message for queue empty and overflow conditions.

```
#include <stdio.h>
#define SIZE 3
int queue[SIZE];
int front = -1, rear = -1;

void insert() {
   int x;
   if ((rear + 1) % SIZE == front) {
      printf("Queue Overflow!\n");
      return;
   }
   printf("Enter value: ");
   scanf("%d", &x);
   if (front == -1) front = rear = 0;
   else rear = (rear + 1) % SIZE;
   queue[rear] = x;
   printf("%d inserted.\n", x);
}

void delete() {
   if (front == -1) {
      printf("Queue Underflow!\n");
      return;
   }
   printf("%d deleted.\n", queue[front]);
   if (front == rear) front = rear = -1;
   else front = (front + 1) % SIZE;
}

void display() {
   if (front == -1) {
      printf("Queue is empty.\n");
      return;
   }

   printf("Queue: ");
   int i = front;
   while (1) {
      printf("%d ", queue[i]);
      if (i == rear) break;
      i = (i + 1) % SIZE;
   }
   printf("\n");
}

int main() {
   int choice;
```

```
    printf("---Circular Queue---");
       while (1) {
          printf("\n1.Insert 2.Delete 3.Display 4.Exit\n Enter 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!\n");
          }
       }
    }
```

Output:



```
1.Insert 2.Delete 3.Display 4.Exit
 Enter Choice: 3
Queue: 30 40

1.Insert 2.Delete 3.Display 4.Exit
 Enter Choice: 2
30 deleted.

1.Insert 2.Delete 3.Display 4.Exit
 Enter Choice: 2
40 deleted.

1.Insert 2.Delete 3.Display 4.Exit
 Enter Choice: 2
Queue Underflow!

1.Insert 2.Delete 3.Display 4.Exit
 Enter Choice: 3
Queue is empty.

1.Insert 2.Delete 3.Display 4.Exit
 Enter Choice: 4

Process returned 0 (0x0)   execution time : 80.723 s
Press any key to continue.
```

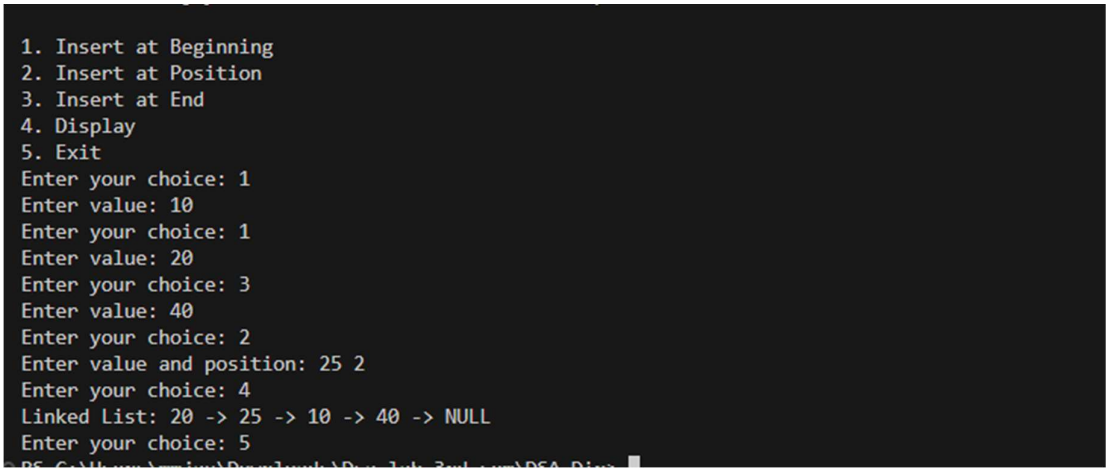
Lab Program 4
WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.

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

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

struct node *head = NULL, *ptr, *temp;

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

// Insert at beginning
void insertAtBeginning(int value) {
   ptr = createNode(value);
   ptr->next = head;
   head = ptr;
}

// Insert at end
void insertAtEnd(int value) {
   ptr = createNode(value);
   if (head == NULL)
      head = ptr;
   else {
      temp = head;
      while (temp->next != NULL)
         temp = temp->next;
      temp->next = ptr;
   }
}

// Insert at any position
void insertAtPosition(int value, int pos) {
   int i;
   ptr = createNode(value);
   if (pos == 1) {
      ptr->next = head;
      head = ptr;
```

```
        return;
    }
    temp = head;
    for (i = 1; i < pos - 1 && temp != NULL; i++)
        temp = temp->next;
    if (temp == NULL)
        printf("Position out of range!\n");
    else {
        ptr->next = temp->next;
        temp->next = ptr;
    }
}

// Display the list
void display() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Main function
int main() {
    int choice, value, pos;
printf("\n1. Insert at Beginning\n2. Insert at Position\n3. Insert at End\n4. Display\n5.
Exit\n");
    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter value: ");
            scanf("%d", &value);
            insertAtBeginning(value);
            break;
        case 2:
            printf("Enter value and position: ");
            scanf("%d %d", &value, &pos);
            insertAtPosition(value, pos);
            break;
        case 3:
            printf("Enter value: ");
            scanf("%d", &value);
```

```
            insertAtEnd(value);
            break;
        case 4:
            display();
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

Output:



Lab Program 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.

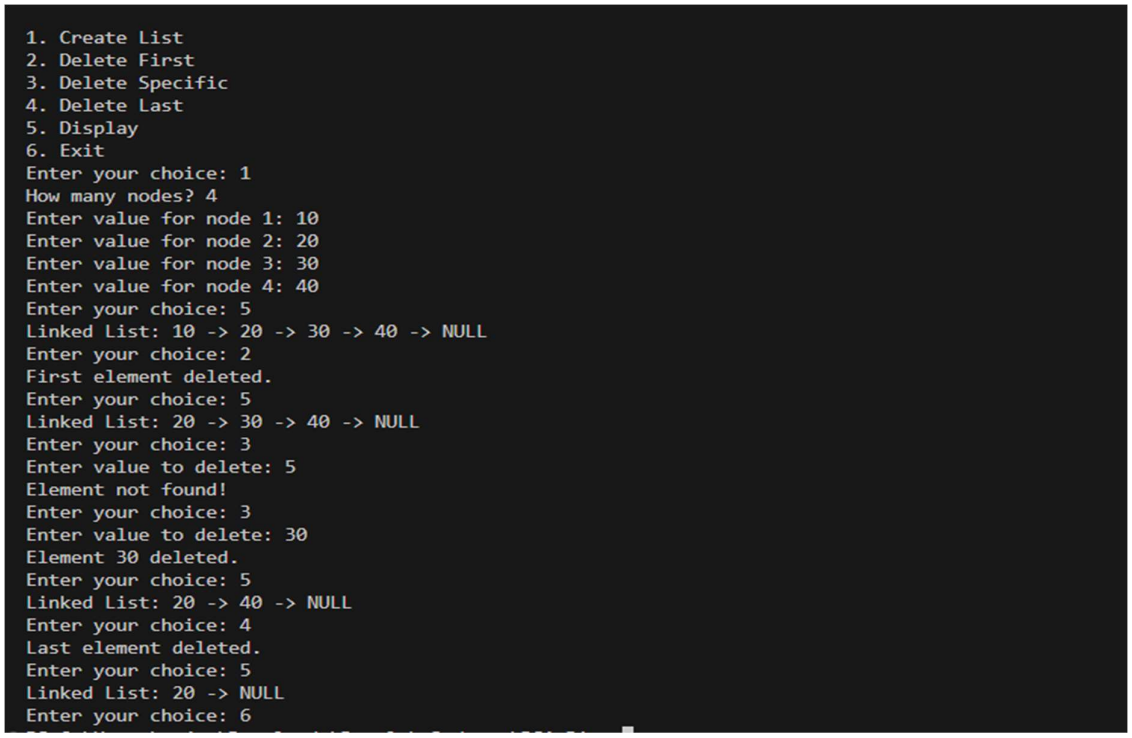
```
#include <stdio.h>
#include <stdlib.h>

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

    struct node *head = NULL, *ptr, *temp;

    // Create node function
    struct node* createNode(int value) {
       ptr = (struct node*)malloc(sizeof(struct node));
       ptr->data = value;
       ptr->next = NULL;
       return ptr;
    }

    // Create linked list
    void createList(int n) {
       int value, i;
       if (n <= 0) return;

       printf("Enter value for node 1: ");
       scanf("%d", &value);
       head = createNode(value);
       temp = head;

       for (i = 2; i <= n; i++) {
          printf("Enter value for node %d: ", i);
          scanf("%d", &value);
          ptr = createNode(value);
          temp->next = ptr;
          temp = temp->next;
       }
    }

    // Delete first element
    void deleteFirst() {
       if (head == NULL) {
          printf("List is empty!\n");
          return;
       }
```

```
    ptr = head;
    head = head->next;
    free(ptr);
    printf("First element deleted.\n");
}

// Delete last element
void deleteLast() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    if (head->next == NULL) {
        free(head);
        head = NULL;
        printf("Last element deleted.\n");
        return;
    }
    temp = head;
    while (temp->next->next != NULL)
        temp = temp->next;
    free(temp->next);
    temp->next = NULL;
    printf("Last element deleted.\n");
}

// Delete specified element by value
void deleteSpecific(int value) {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    ptr = head;
    if (head->data == value) {
        head = head->next;
        free(ptr);
        printf("Element %d deleted.\n", value);
        return;
    }
    while (ptr->next != NULL && ptr->next->data != value)
        ptr = ptr->next;

    if (ptr->next == NULL)
        printf("Element not found!\n");
    else {
        temp = ptr->next;
        ptr->next = temp->next;
        free(temp);
        printf("Element %d deleted.\n", value);
    }
}
```

```
// Display linked list
void display() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Main function
int main() {
    int choice, n, value;
printf("\n1. Create List\n2. Delete First\n3. Delete Specific\n4. Delete Last\n5. Display\n6.
Exit\n");
    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("How many nodes? ");
            scanf("%d", &n);
            createList(n);
            break;
        case 2:
            deleteFirst();
            break;
        case 3:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            deleteSpecific(value);
            break;
        case 4:
            deleteLast();
            break;
        case 5:
            display();
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
        }
    }
```

```
    return 0;
}
```

Output:



Lab Program 6

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* create(int x) {
    struct Node* n = (struct Node*)malloc(sizeof(struct Node));
    n->data = x;
    n->next = NULL;
    return n;
}

struct Node* insertEnd(struct Node* head, int x) {
    struct Node* n = create(x);
    if (!head) return n;

    struct Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
    return head;
}

void printList(struct Node* head) {
    printf("List: ");
    while (head) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

void sortList(struct Node* head) {
    struct Node *i, *j;
    int temp;

    for (i = head; i && i->next; i = i->next)
        for (j = i->next; j; j = j->next)
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }

    printf("After Sorting: ");
```

```
    printList(head);
}

struct Node* reverseList(struct Node* head) {
    struct Node *prev = NULL, *cur = head, *next;

    while (cur) {
        next = cur->next;
        cur->next = prev;
        prev = cur;
        cur = next;
    }

    printf("After Reversing: ");
    printList(prev);
    return prev;
}

struct Node* concatList(struct Node* a, struct Node* b) {
    if (!a) return b;
    struct Node* t = a;

    while (t->next) t = t->next;
    t->next = b;

    printf("After Concatenation: ");
    printList(a);

    return a;
}

int main() {
    struct Node *L1 = NULL, *L2 = NULL;
    int ch, val;
      printf("\n--- MENU ---\n");
        printf("1. Insert into List 1\n");
        printf("2. Insert into List 2\n");
        printf("3. Sort List 1\n");
        printf("4. Reverse List 1\n");
        printf("5. Concatenate List1 + List2\n");
        printf("6. Print List 1\n");
        printf("7. Print List 2\n");
        printf("8. Exit\n");
    while (1) {

        printf("Choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                scanf("%d", &val);
```

```
            L1 = insertEnd(L1, val);
            break;
        case 2:
            scanf("%d", &val);
            L2 = insertEnd(L2, val);
            break;
        case 3:
            sortList(L1);
            break;
        case 4:
            L1 = reverseList(L1);
            break;
        case 5:
            L1 = concatList(L1, L2);
            break;
        case 6:
            printList(L1);
            break;
        case 7:
            printList(L2);
            break;
        case 8:
            exit(0);
        default:
            printf("Invalid Choice!\n");
    }
  }
}
```

Output:

```
--- MENU ---
1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate List1 + List2
6. Print List 1
7. Print List 2
8. Exit
Choice: 1
10
Choice: 1
20
Choice: 1
30
Choice: 2
60
Choice: 2
50
Choice: 6
List: 10 -> 20 -> 30 -> NULL
Choice: 7
List: 60 -> 50 -> NULL
Choice: 3
After Sorting: List: 10 -> 20 -> 30 -> NULL
Choice: 4
After Reversing: List: 30 -> 20 -> 10 -> NULL
Choice: 5
After Concatenation: List: 30 -> 20 -> 10 -> 60 -> 50 -> NULL
Choice: 8
```

6(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* create(int x) {
   struct Node* n = (struct Node*)malloc(sizeof(struct Node));
   n->data = x;
   n->next = NULL;
   return n;
}

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

struct Node* push(struct Node* top, int x) {
   struct Node* n = create(x);
   n->next = top;
   return n;
}

struct Node* pop(struct Node* top) {
   if (!top) { printf("Stack Underflow!\n"); return NULL; }
   struct Node* t = top;
   top = top->next;
   free(t);
   return top;
}

struct Node* enqueue(struct Node* front, int x) {
   struct Node* n = create(x);
   if (!front) return n;
   struct Node* t = front;
   while (t->next) t = t->next;
   t->next = n;
   return front;
}

struct Node* dequeue(struct Node* front) {
   if (!front) { printf("Queue Underflow!\n"); return NULL; }
```

```
    struct Node* t = front;
    front = front->next;
    free(t);
    return front;
}

int main() {
    struct Node *stack = NULL, *queue = NULL;
    int ch, val;
     printf("\n1.Push  2.Pop  3.Display Stack\n");
    printf("4.Enqueue  5.Dequeue  6.Display Queue\n7.Exit\n");
    while (1) {

        printf("Choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1: scanf("%d",&val); stack = push(stack,val); print(stack); break;
            case 2: stack = pop(stack); print(stack); break;
            case 3: print(stack); break;

            case 4: scanf("%d",&val); queue = enqueue(queue,val); print(queue); break;
            case 5: queue = dequeue(queue); print(queue); break;
            case 6: print(queue); break;

            case 7: exit(0);
        }
    }
}
```

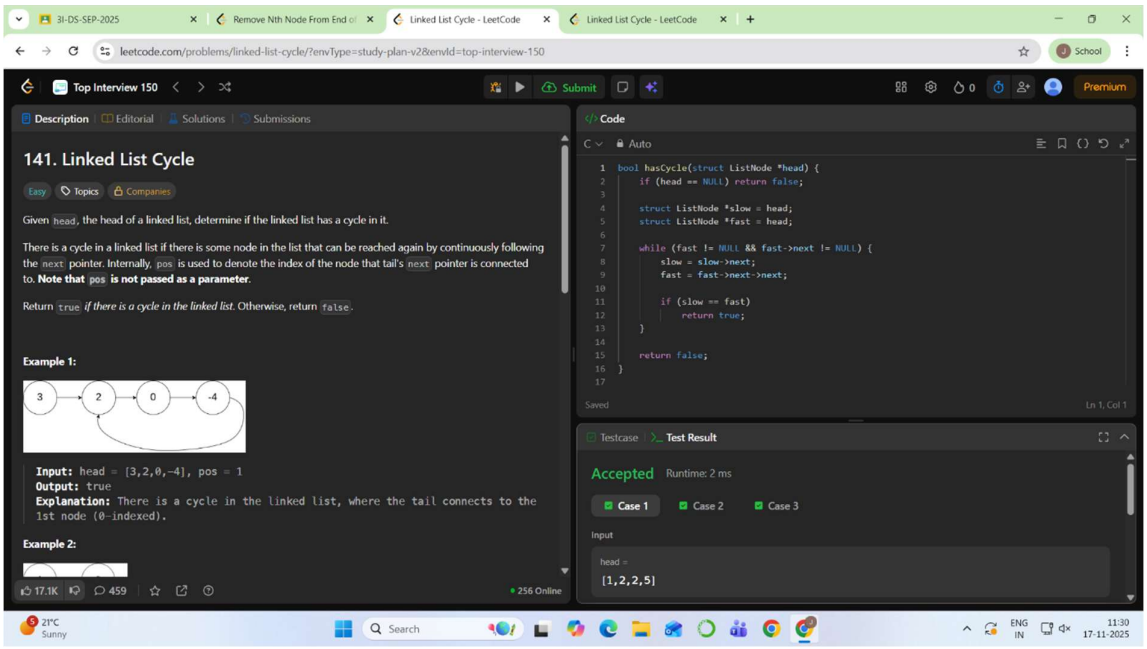
Output:

```
1.Push  2.Pop  3.Display Stack
4.Enqueue  5.Dequeue  6.Display Queue
7.Exit
Choice: 1
10
10 -> NULL
Choice: 1
20
20 -> 10 -> NULL
Choice: 1
30
30 -> 20 -> 10 -> NULL
Choice: 3
30 -> 20 -> 10 -> NULL
Choice: 4
40
40 -> NULL
Choice: 4
50
40 -> 50 -> NULL
Choice: 4
60
40 -> 50 -> 60 -> NULL
Choice: 2
20 -> 10 -> NULL
Choice: 5
50 -> 60 -> NULL
Choice: 5
60 -> NULL
Choice: 7
```

LEETCODE-141

Linked List Cycle

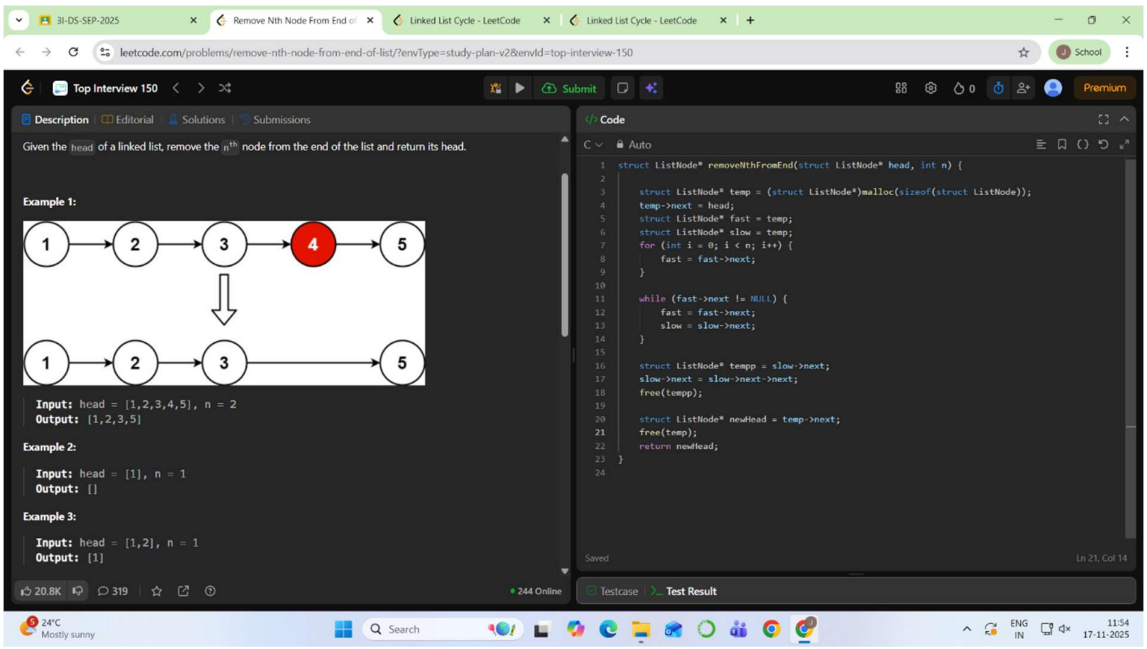
```
bool hasCycle(struct ListNode *head) {
    if (head == NULL) return false;

    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;
}
```



LEETCODE-19

REMOVE NTH NODE FROM END OF THE LIST

```
struct ListNode* removeNthFromEnd(struct ListNode* head, int n) {

    struct ListNode* dummy = (struct ListNode*)malloc(sizeof(struct ListNode));
    dummy->next = head;
    struct ListNode* fast = dummy;
    struct ListNode* slow = dummy;
    for (int i = 0; i < n; i++) {
        fast = fast->next;
    }

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

    struct ListNode* temp = slow->next;
    slow->next = slow->next->next;
    free(temp);

    struct ListNode* newHead = dummy->next;
    free(dummy);
    return newHead;
}
```



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

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

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

struct Node *head = NULL;

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

// a) Create DLL — insert at end
void createDLL() {
   int n, value;
   printf("Enter number of nodes: ");
   scanf("%d", &n);

   for (int i = 0; i < n; i++) {
      printf("Enter value for node %d: ", i + 1);
      scanf("%d", &value);

      struct Node *newNode = createNode(value);

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

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

// b) Insert LEFT of a node
void insertLeft() {
    int key, value;
    printf("Enter key (value of node): ");
    scanf("%d", &key);
    printf("Enter new value to insert: ");
    scanf("%d", &value);

    struct Node *temp = head;

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

    if (!temp) {
        printf("Key not found.\n");
        return;
    }

    struct Node *newNode = createNode(value);

    if (temp == head) {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    } else {
        newNode->next = temp;
        newNode->prev = temp->prev;
        temp->prev->next = newNode;
        temp->prev = newNode;
    }
}

// c) Delete a node by value
void deleteValue() {
    int key;
```

```
    printf("Enter value to delete: ");
    scanf("%d", &key);

    struct Node *temp = head;

    while (temp && temp->data != key)
        temp = temp->next;
    if (!temp) {
        printf("Value not found.\n");
        return;
    }
    if (temp == head)
        head = head->next;
    if (temp->next)
        temp->next->prev = temp->prev;
    if (temp->prev)
        temp->prev->next = temp->next;
    free(temp);
    printf("Node deleted.\n");
}

// d) Display DLL
void display() {
    struct Node *temp = head;
    if (!head) {
        printf("List is empty.\n");
        return;
    }
    printf("List: ");
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice;
printf("\n--- MENU ---\n");
        printf("1. Create Doubly Linked List\n");
        printf("2. Insert Left of Node\n");
        printf("3. Delete Node by Value\n");
```

```
        printf("4. Display List\n");
        printf("5. Exit\n");
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: createDLL(); break;
            case 2: insertLeft(); break;
            case 3: deleteValue(); break;
            case 4: display(); break;
            case 5: exit(0);
            default: printf("Invalid choice!\n");
        }
    }
}
```

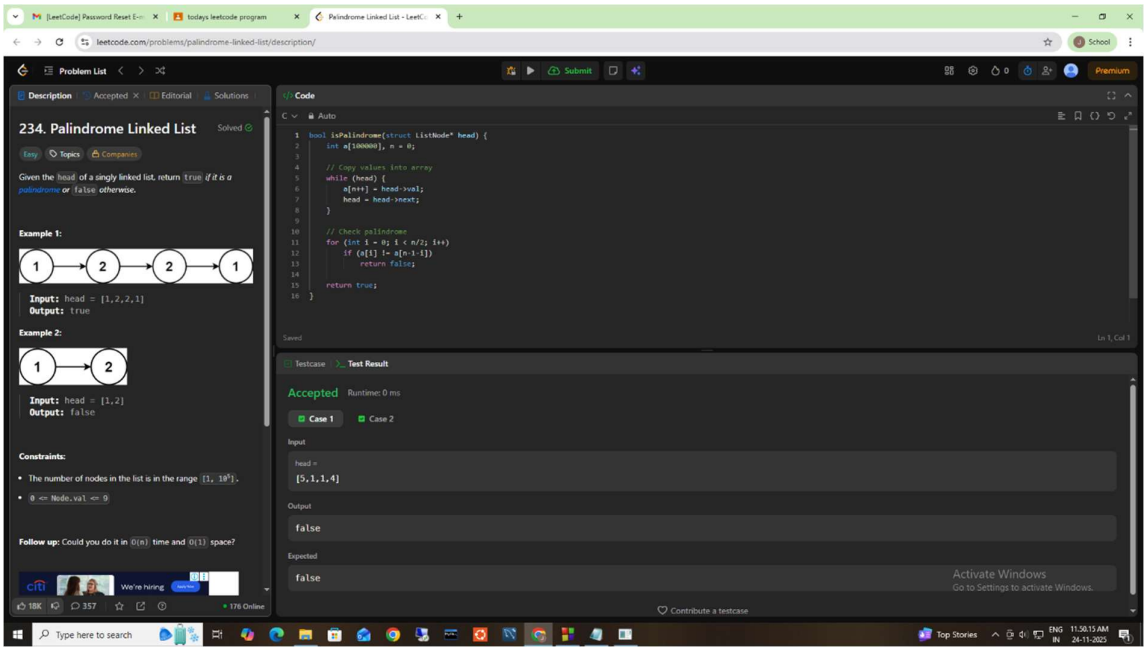
Output:

```
--- MENU ---
1. Create Doubly Linked List
2. Insert Left of Node
3. Delete Node by Value
4. Display List
5. Exit

Enter your choice: 1
Enter number of nodes: 3
Enter value for node 1: 10
Enter value for node 2: 20
Enter value for node 3: 30

Enter your choice: 2
Enter key (value of node): 30
Enter new value to insert: 25

Enter your choice: 4
List: 10 <=>20 <=>25 <=>30 <=>

Enter your choice: 3
Enter value to delete: 30
Node deleted.

Enter your choice: 4
List: 10 <=>20 <=>25 <=>

Enter your choice: 5
```

LEETCODE-234

PALINDROME LINKED LIST

```
bool isPalindrome(struct ListNode* head) {
    int a[100000], n = 0;

    // Copy values into array
    while (head) {
        a[n++] = head->val;
        head = head->next;
    }

    // Check palindrome
    for (int i = 0; i < n/2; i++)
        if (a[i] != a[n-1-i])
            return false;

    return true;
}
```



Lab Program 8

8(a)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 for a tree node
struct node {
    int data;
    struct node *left, *right;
};

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

// Function to insert a value into the BST
struct node* insert(struct node* root, int value) {
    if (root == NULL) {
        root = createNode(value);
    }
    else if (value < root->data) {
        root->left = insert(root->left, value);
    }
    else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}

// Inorder traversal (Left → Root → Right)
void inorder(struct node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
```

```
}

// Preorder traversal (Root → Left → Right)
void preorder(struct node* root) {
   if (root != NULL) {
      printf("%d ", root->data);
      preorder(root->left);
      preorder(root->right);
   }
}

// Postorder traversal (Left → Right → Root)
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, value;
    printf("\n Binary Search Tree Menu \n 1. Insert element\n 2. Display Inorder
Traversal\n 3. Display Preorder Traversal\n 4. Display Postorder Traversal\n 5. Exit\n");

   while (1) {

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

      switch (choice) {
         case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            root = insert(root, value);
            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:
                printf("Exiting program.\n");
                exit(0);

            default:
                printf("Invalid choice, try again.\n");
        }
    }

    return 0;
}
```

Output:

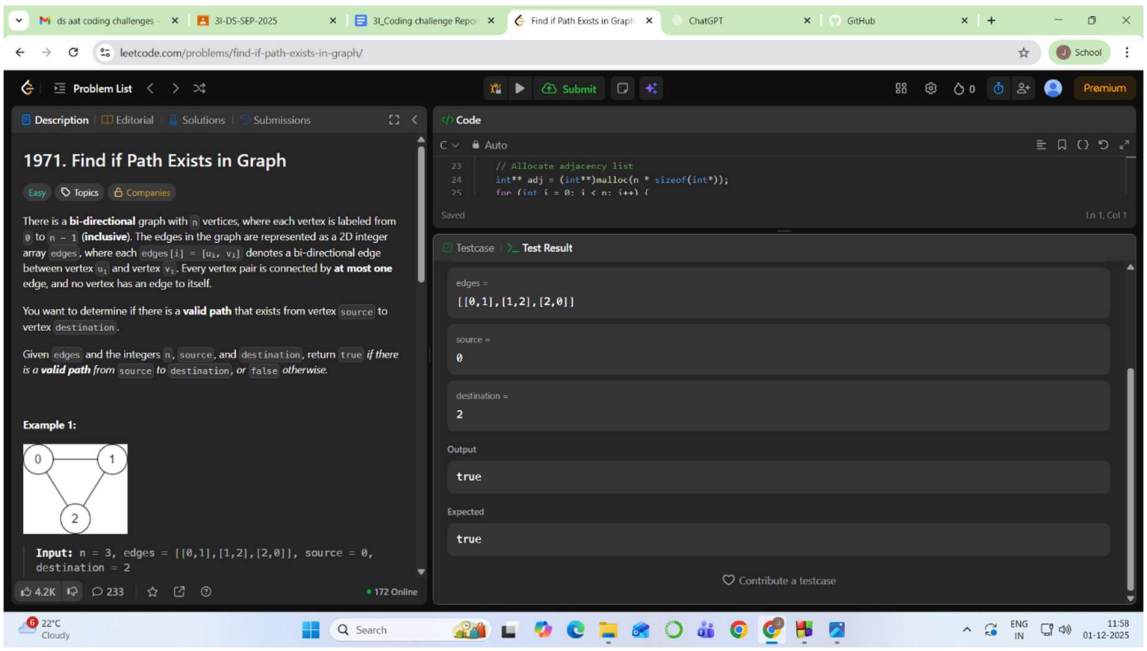
```
 Binary Search Tree Menu
 1. Insert element
 2. Display Inorder Traversal
 3. Display Preorder Traversal
 4. Display Postorder Traversal
 5. Exit
Enter your choice: 1
Enter value to insert: 20
Enter your choice: 1
Enter value to insert: 4
Enter your choice: 1
Enter value to insert: 79
Enter your choice: 1
Enter value to insert: 38
Enter your choice: 1
Enter value to insert: 28
Enter your choice: 1
Enter value to insert: 80
Enter your choice: 1
Enter value to insert: 45
Enter your choice: 2
Inorder Traversal: 4 20 28 38 45 79 80
Enter your choice: 3
Preorder Traversal: 20 4 79 38 28 45 80
Enter your choice: 4
Postorder Traversal: 4 28 45 38 80 79 20
Enter your choice: 5
Exiting program.
```

LEETCODE-1971

Find if the path exists in graph

```
bool validPath(int n, int** edges, int edgesSize, int* edgesColSize,
               int source, int destination) {

    if (source == destination) return true;

    int* deg = calloc(n, sizeof(int));
    for (int i = 0; i < edgesSize; i++) {
        deg[edges[i][0]]++;
        deg[edges[i][1]]++;
    }

    int** g = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++)
        g[i] = malloc(deg[i] * sizeof(int));

    int* d = calloc(n, sizeof(int));
    for (int i = 0; i < edgesSize; i++) {
        int u = edges[i][0], v = edges[i][1];
        g[u][d[u]++] = v;
        g[v][d[v]++] = u;
    }

    bool* vis = calloc(n, sizeof(bool));
    int* q = malloc(n * sizeof(int));
    int f = 0, r = 0;

    q[r++] = source;
    vis[source] = true;

    while (f < r) {
        int u = q[f++];
        if (u == destination) return true;
        for (int i = 0; i < deg[u]; i++) {
            int v = g[u][i];
            if (!vis[v]) {
                vis[v] = true;
                q[r++] = v;
            }
        }
    }
```

```
    return false;
}
```



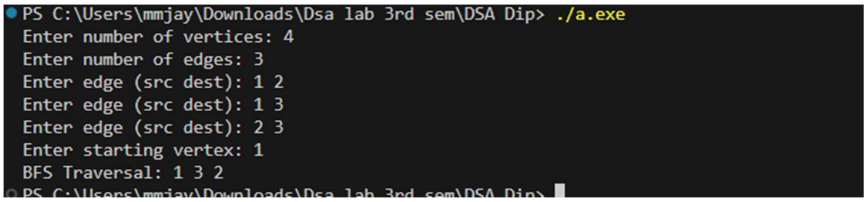
Lab Program 9

(A) Write a program to traverse a graph using BFS method.

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

#define MAX 20

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

struct Node* adj[MAX];
int visited[MAX];
int queue[MAX], front = 0, rear = 0;
int n;

void addEdge(int src, int dest) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->vertex = dest;
   newNode->next = adj[src];
   adj[src] = newNode;
}

void bfs(int start) {
   int v;
   queue[rear++] = start;
   visited[start] = 1;

   printf("BFS Traversal: ");
   while (front < rear) {
      v = queue[front++];
      printf("%d ", v);

      struct Node* temp = adj[v];
      while (temp != NULL) {
         if (!visited[temp->vertex]) {
            queue[rear++] = temp->vertex;
            visited[temp->vertex] = 1;
         }
         temp = temp->next;
      }
   }
}

int main() {
   int edges, src, dest, start;

   printf("Enter number of vertices: ");
```

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

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

    for (int i = 0; i < edges; i++) {
        printf("Enter edge (src dest): ");
        scanf("%d %d", &src, &dest);
        addEdge(src, dest);
        addEdge(dest, src);
    }

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

    bfs(start);

    return 0;
}
```

Output:

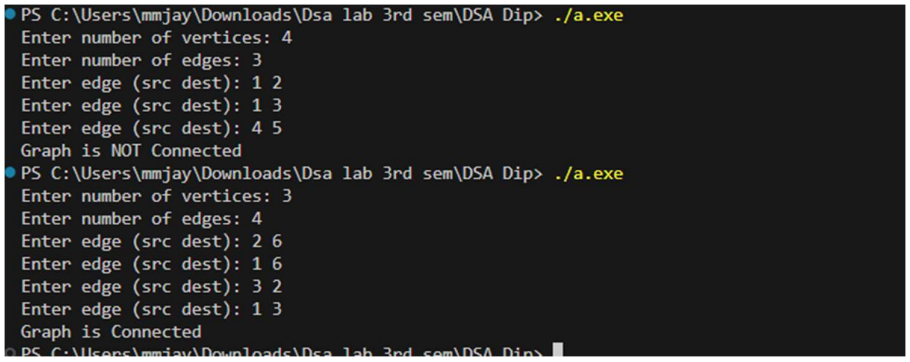


B) Program to Check Whether Graph is Connected Using DFS (Adjacency List)
Program Code:

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

#define MAX 20

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

struct Node* adj[MAX];
int visited[MAX];
int n;

void addEdge(int src, int dest) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->vertex = dest;
   newNode->next = adj[src];
   adj[src] = newNode;
}

void dfs(int v) {
   visited[v] = 1;

   struct Node* temp = adj[v];
   while (temp != NULL) {
      if (!visited[temp->vertex])
         dfs(temp->vertex);
      temp = temp->next;
   }
}

int main() {
   int edges, src, dest;

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

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

   for (int i = 0; i < edges; i++) {
      printf("Enter edge (src dest): ");
      scanf("%d %d", &src, &dest);
      addEdge(src, dest);
      addEdge(dest, src);
   }
```

```
    dfs(1);

    for (int i = 1; i <= n; i++) {
        if (!visited[i]) {
            printf("Graph is NOT Connected\n");
            return 0;
        }
    }

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

Output:



Lab Program 10

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.

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

```
#include <stdio.h>
#define MAX 100

int hashTable[MAX];
int m;   // size of hash table

// Function to initialize hash table
void initialize()
{
   for (int i = 0; i < m; i++)
      hashTable[i] = -1;   // -1 indicates empty location
}

// Hash function H(K) = K mod m
int hashFunction(int key)
{
   return key % m;
}

// Function to insert a key using Linear Probing
void insert(int key)
{
   int index = hashFunction(key);

   // Linear probing in case of collision
   while (hashTable[index] != -1)
   {
      index = (index + 1) % m;
   }

   hashTable[index] = key;
   printf("Key %d inserted at address %d\n", key, index);
}
```

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

int main()
{
   int n, key;

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

   initialize();

   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;
}
```

Output:

```
PS C:\Users\mmjay\Downloads\Dsa lab 3rd sem\DSA Dip> ./a.exe
Enter size of hash table (m): 10
Enter number of employee records (N): 5
Enter 5 employee keys (4-digit):
1127
Key 1127 inserted at address 7
1128
Key 1128 inserted at address 8
1135
Key 1135 inserted at address 5
1179
Key 1179 inserted at address 9
1569
Key 1569 inserted at address 0

Hash Table:
Address Key
0       1569
1       EMPTY
2       EMPTY
3       EMPTY
4       EMPTY
5       1135
6       EMPTY
7       1127
8       1128
9       1179
```