VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Dama Yohitesh Naveen Sai

(1BM23CS085)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU) BENGALURU-560019
September 2024-January 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by **Dama Yohitesh Naveen Sai(1BM23CS085)**, 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 202425. 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. Selva Kumar S	**Dr. Kavitha Sooda**
Associate Professor	Professor and Head
Department Of CSE	Of department CSE
BMSCE, Bengaluru	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.

GitHub Link: https://github.com/Yohitesh/DS-lab

Lab program 1

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 100

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

void push(int value) {
    if (top == MAX - 1) {
        printf("Stack Overflow! Cannot push %d.\n", value);
    } else {
        top++;
        stack[top] = value;
        printf("%d pushed onto the stack.\n", value);
    }
}

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

void display() {
    if (top == -1)
    {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements are: ");
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}
```

```
int main() {
    int choice, value;

    while (1) {
        printf("\nStack
        Operations:\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:
                printf("Enter the value to push:
                "); scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

Output:

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 4
4 pushed onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
4 popped from the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack is empty.

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

Lab Program 2

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 c) {

    if (top == MAX - 1) {

        printf("Stack

        Overflow!\n");

    } else {

        stack[++top] = c;

    }

}

char pop() {

    if (top == -1) {

        printf("Stack Underflow!\n");
```

```
        return -1;

    } else {
         return stack[top--];

    }

}

char peek() {

    if (top == -1) {

        return -1;

    } else {

        return stack[top];

    }

}

int precedence(char op) {

    switch (op) {

        case '+':

        case '-':

            return

        1; case '*':

        case '/':

            return

        2; default:
```

```
        return 0;
    }
void infixToPostfix(char* infix) {
    char postfix[MAX];
    int i = 0, j =
    0; char c;
    while ((c = infix[i++]) != '\0')
        { if (isalnum(c)) { //
        Operand
            postfix[j++] = c;
        } else if (c == '(') {
            push(c);
        } else if (c == ')') {
            while (peek() != '(') {
                postfix[j++] = pop();
            }
            pop(); // Remove '('
        } else { // Operator
            while (top != -1 && precedence(peek()) >= precedence(c))
            postfix[j++] = pop();
```

```
            }

            push(c);
}

        while (top != -1) {

            postfix[j++] = pop();

        }

        postfix[j] = '\0';

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

    }

    int main() {

        char infix[MAX];

        printf("Enter a valid infix expression:

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

        infixToPostfix(infix);

        return 0;

  }
```
Output:

```
Enter a valid parenthesized infix expression: ((A+B)*(C-D))
Postfix expression: AB+CD-*
```

Lab Program 3

3a. WAP to simulate the working of a queue of integers using an array.

Provide the following operations: Insert, Delete, Display

The program should print appropriate messages for queue empty and queue overflow conditions.

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

int queue[MAX];
int front = -1, rear = -1;
void insert(int value) {
    if ((rear + 1) % MAX == front) {
        printf("Queue Overflow! Cannot insert %d.\n", value);
    } else {
        if (front == -1) {
            front = 0;
        }
        rear = (rear + 1) % MAX;
        queue[rear] = value;
        printf("%d inserted into the queue.\n", value);
    }
}
void delete() {
    if (front == -1) {
        printf("Queue Underflow! Cannot delete.\n");
    } else {
        printf("%d deleted from the queue.\n", queue[front]);
        if (front == rear) {
            front = rear = -1; // Reset the queue
```

```
        } else {
            front = (front + 1) % MAX;
        }
    }
}
void display() {
    if (front == -1) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue elements are: ");
        int i = front;
        while (1) {
            printf("%d ", queue[i]);
            if (i == rear) {
                break;
            }
            i = (i + 1) % MAX;
        }
        printf("\n");
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nQueue Operations:\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:
                printf("Enter the value to insert:
                "); scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete()
                ; break;
            case 3:
                display()
                ; break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

Output:

```
Enter no of elements
3

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 4
Inserted 4

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 5
Inserted 5

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 4

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 5

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue is underflow
```

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

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

 int queue[MAX];
 int front = -1, rear = -1;

 // Function to insert an element into the circular queue
 void insert(int value) {
     if ((rear + 1) % MAX == front) {
         printf("Queue Overflow! Cannot insert %d.\n", value);
     } else {
         if (front == -1) {
             front = 0;
         }
         rear = (rear + 1) % MAX;
         queue[rear] = value;
         printf("%d inserted into the queue.\n", value);
     }
 }
```

```c
// Function to delete an element from the circular queue
void delete() {

    if (front == -1) {

        printf("Queue Underflow! Cannot delete.\n");

    } else {

        printf("%d deleted from the queue.\n", queue[front]);

        if (front == rear) {

            front = rear = -1; // Reset the queue

        } else {

            front = (front + 1) % MAX;

        }

    }

}


// Function to display the elements of the circular queue

void display() {

    if (front == -1) {

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

    } else {

        printf("Queue elements are: ");

        int i = front;

        while (1) {

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

            if (i == rear) {

                break;

            }
```

```
            i = (i + 1) % MAX;
        }
        printf("\n");
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nCircular Queue Operations:\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:
                printf("Enter the value to insert:
                "); scanf("%d", &value);
                insert(value);
                break;
            case 2:
```

```
            delete()
            ; break;
        case 3:
            display()
            ; break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
      }
    }
}
```

Output:

```
Enter no of elements
4

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 4
Inserted 4

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 8
Inserted 8

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 7
Inserted 7

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 0
Inserted 0

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 6
Queue is overflow

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 4

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 8

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 7

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 0

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue is underflow
```

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.

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;

void createLinkedList(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); newNode->data = value;
    newNode->next = NULL;
    head = newNode;
    printf("Linked list created with value %d.\n", value);
}
void insertAtBeginning(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    head = newNode;
    printf("%d inserted at the beginning.\n", value);
}
void insertAtPosition(int value, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (position == 1) {
        newNode->next = head;
        head = newNode;
        printf("%d inserted at position %d.\n", value, position);
        return;
    }

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

```
    if (temp == NULL) {
        printf("Invalid position!\n");
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
        printf("%d inserted at position %d.\n", value, position);
    }
}
void insertAtEnd(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        printf("%d inserted at the end.\n", value);
        return;
    }

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

    temp->next = newNode;
    printf("%d inserted at the end.\n", value);
}
void displayLinkedList() {
    if (head == NULL) {
        printf("Linked list is empty.\n");
        return;
    }

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

int main() {
    int choice, value, position;

    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Create Linked List\n");
        printf("2. Insert at Beginning\n");
```

```
        printf("3. Insert at Any Position\n");
        printf("4. Insert at End\n");
        printf("5. Display Linked List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value for the first node: ");
                scanf("%d", &value);
                createLinkedList(value);
                break;
            case 2:
                printf("Enter the value to insert at the beginning: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 3:
                printf("Enter the value to insert: ");
                scanf("%d", &value); printf("Enter the position: "); scanf("%d", &position);
                insertAtPosition(value, position); break;
            case 4:
                printf("Enter the value to insert at the end: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 5:
                displayLinkedList();
                break;
            case 6:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

OUTPUT:

```
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Delete at beginning
5. Delete at end
6. Delete a value
7. Display
8. Exit
Enter choice: 1
Enter the element: 23
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Delete at beginning
5. Delete at end
6. Delete a value
7. Display
8. Exit
Enter choice: 1
Enter the element: 89
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Delete at beginning
5. Delete at end
6. Delete a value
7. Display
8. Exit
Enter choice: 2
Enter the element: 97
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Delete at beginning
5. Delete at end
6. Delete a value
7. Display
8. Exit
Enter choice: 3
Enter the element and position: 12
2
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Delete at beginning
5. Delete at end
6. Delete a value
7. Display
8. Exit
Enter choice: 7
89 12 23 97
```

Lab Program 5

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* head = NULL;

struct Node* createNode(int value) {

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

    newNode->data = value;

    newNode->next = NULL;

    return newNode;

}

void insertAtEnd(int value) {

    struct Node* newNode = createNode(value);

    if (head == NULL) {

        head = newNode;
```

```
        return;
    }
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}
void displayLinkedList(struct Node* headRef) {
    if (headRef == NULL) {
        printf("Linked list is empty.\n");
        return;
    }
    struct Node* temp = headRef;
    printf("Linked list contents: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
```

```
    printf("NULL\n");
}
void sortLinkedList() {
    if (head == NULL || head->next == NULL) {
        return;
    }
    struct Node* i;
    struct Node* j;
    for (i = head; i->next != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                int temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
    printf("Linked list sorted.\n");
}
```

```
}

void reverseLinkedList() {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    printf("Linked list reversed.\n");
}

struct Node* concatenateLists(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) {
        return head2;
    }
    if (head2 == NULL) {
        return head1;
```

```
    }

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

int main() {
    int choice, value;
    struct Node* secondList = NULL;

    while (1) {
        printf("\nSingle Linked List Operations:\n");
        printf("1. Insert at End\n");
        printf("2. Display Linked List\n");
        printf("3. Sort Linked List\n");
```

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

printf("5. Concatenate with Another List\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

    case 1:

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

        scanf("%d", &value);

        insertAtEnd(value);

        break;

    case 2:

        displayLinkedList(head);

        break;

    case 3:

        sortLinkedList();

        break;

    case 4:

        reverseLinkedList();
```

```
    break;
case 5:
    printf("Enter the number of elements for the second list: ");
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter value %d: ", i + 1);
        scanf("%d", &value);
        struct Node* newNode = createNode(value);
        if (secondList == NULL) {
            secondList = newNode;
        } else {
            struct Node* temp = secondList;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }
```

```c
                head = concatenateLists(head, secondList);
                printf("Lists concatenated.\n");
                break;
            case 6:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

OUTPUT:

```
1. Insert at End (List 1)
2. Insert at End (List 2)
3. Sort List 1
4. Reverse List 1
5. Concatenate List 2 to List 1
6. Display List 1
7. Display List 2
8. Exit
Enter your choice: 1
Enter data: 20
1. Insert at End (List 1)
2. Insert at End (List 2)
3. Sort List 1
4. Reverse List 1
5. Concatenate List 2 to List 1
6. Display List 1
7. Display List 2
8. Exit
Enter your choice: 1
Enter data: 10
1. Insert at End (List 1)
2. Insert at End (List 2)
3. Sort List 1
4. Reverse List 1
5. Concatenate List 2 to List 1
6. Display List 1
7. Display List 2
8. Exit
Enter your choice: 1
Enter data: 50
1. Insert at End (List 1)
2. Insert at End (List 2)
3. Sort List 1
4. Reverse List 1
5. Concatenate List 2 to List 1
6. Display List 1
7. Display List 2
8. Exit
Enter your choice: 2
Enter data: 40
1. Insert at End (List 1)
2. Insert at End (List 2)
3. Sort List 1
4. Reverse List 1
5. Concatenate List 2 to List 1
6. Display List 1
7. Display List 2
8. Exit
Enter your choice: 2
Enter data: 30
1. Insert at End (List 1)
2. Insert at End (List 2)
3. Sort List 1
```

LAB PROGRAM 6

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* head = NULL;
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
    Node)); newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
void push(int value) {
    struct Node* newNode =
    createNode(value); newNode->next = head;
    head = newNode;
    printf("%d pushed to stack.\n", value);
}

void pop() {
    if (head == NULL) {
        printf("Stack underflow.\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    printf("%d popped from stack.\n", temp->data);
    free(temp);
}

void displayStack() {
    if (head == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    struct   Node*   temp   =
    head;          printf("Stack
    contents:  ");  while  (temp
```

```
    != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
struct Node* tail = NULL; // Tail pointer for queue
void enqueue(int value) {
    struct Node* newNode = createNode(value);
    if (tail == NULL) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
    printf("%d enqueued to queue.\n", value);
}

void dequeue() {
    if (head == NULL) {
        printf("Queue underflow.\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    if (head == NULL)
        { tail = NULL;
    }
    printf("%d dequeued from queue.\n", temp->data);
    free(temp);
}

void displayQueue() {
    if (head == NULL)
    {
        printf("Queue is empty.\n");
        return;
    }
    struct Node* temp = head;
    printf("Queue     contents:
    "); while (temp != NULL)
    {
        printf("%d          ->          ",
        temp->data);       temp       =
        temp->next;
    }
    printf("NULL\n");
}
```

```
int main() {
    int choice, value;
    while (1) {
        printf("\nSingle Linked List as Stack and Queue:\n");
        printf("1. Push to Stack\n");
        printf("2. Pop from Stack\n");
        printf("3. Display Stack\n");
        printf("4. Enqueue to Queue\n");
        printf("5. Dequeue from Queue\n");
        printf("6. Display Queue\n");
        printf("7. 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:
                displayStack();
                break;
            case 4:
                printf("Enter value to enqueue:
                "); scanf("%d", &value);
                enqueue(value);
                break;
            case 5:
                dequeue();
                break;
            case 6:
                displayQueue();
                break;
            case 7:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
```

Output:

```
1.Push
2. Pop
3. Enqueue
4. Dequeue
5.Display
Enter choice: 1
Enter element :23
1.Push
2. Pop
3. Enqueue
4. Dequeue
5.Display
Enter choice: 1
Enter element :56
1.Push
2. Pop
3. Enqueue
4. Dequeue
5.Display
Enter choice: 2
1.Push
2. Pop
3. Enqueue
4. Dequeue
5.Display
Enter choice: 3
Enter element :45
1.Push
2. Pop
3. Enqueue
4. Dequeue
5.Display
Enter choice: 4
1.Push
2. Pop
3. Enqueue
4. Dequeue
5.Display
Enter choice: 5
23
```

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 list.

```
#include <stdio.h>

#include <stdlib.h>


 struct Node {

    int data;

    struct Node* prev;

    struct Node* next;

};


struct Node* head = NULL;

struct Node* createNode(int value) {

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

    newNode->data = value;

    newNode->prev            =

    NULL; newNode->next =

    NULL; return newNode;

}


void createDoublyLinkedList(int value) {

    struct Node* newNode = createNode(value);

    if (head == NULL) {

        head = newNode;
```

```
        printf("Doubly linked list created with value %d.\n", value);
    } else {
        printf("List already exists.\n");
    }
}

void insertLeft(int target, int value) {
    if (head == NULL) {
        printf("List is empty. Cannot insert.\n");
        return;
    }

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

    if (temp == NULL) {
        printf("Target node with value %d not found.\n", target);
        return;
    }

    struct Node* newNode = createNode(value);
    newNode->next = temp;
    newNode->prev = temp->prev;

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

```
    } else {
        head = newNode;
    }

    temp->prev = newNode;
    printf("%d inserted to the left of %d.\n", value, target);
}

void deleteNode(int value) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }

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

    if (temp == NULL) {
        printf("Node with value %d not found.\n", value);
        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 with value %d deleted.\n", value);
}
void displayList() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = head;
    printf("Doubly linked list contents: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, value, target;

    while (1) {
```

```
printf("\nDoubly Linked List Operations:\n");
printf("1. Create Doubly Linked List\n");
printf("2. Insert to the Left of a Node\n");
printf("3. Delete a Node\n");
printf("4. Display List\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
  case 1:
    printf("Enter value to create the list: ");
    scanf("%d", &value);
    createDoublyLinkedList(value);
    break;
  case 2:
    printf("Enter the target value to insert left of: ");
    scanf("%d", &target);
    printf("Enter the value to insert: ");
    scanf("%d", &value);
    insertLeft(target, value);
    break;
  case 3:
    printf("Enter the value of the node to delete: ");
    scanf("%d", &value);
    deleteNode(value);
    break;
  case 4:
```

```
            displayList();
            break;
        case 5:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }
}
```

OUTPUT:

```
1. Insert new node left of a node
2. Delete node based on value
3. Display
4. Insert at beginning
Enter choice: 4
Enter the element: 23
1. Insert new node left of a node
2. Delete node based on value
3. Display
4. Insert at beginning
Enter choice: 4
Enter the element: 56
1. Insert new node left of a node
2. Delete node based on value
3. Display
4. Insert at beginning
Enter choice: 3
56 23
1. Insert new node left of a node
2. Delete node based on value
3. Display
4. Insert at beginning
Enter choice: 1
Enter the element: 57
Enter the position: 2
1. Insert new node left of a node
2. Delete node based on value
3. Display
4. Insert at beginning
Enter choice: 2
Enter the element: 57
1. Insert new node left of a node
2. Delete node based on value
3. Display
4. Insert at beginning
Enter choice: 3
56 23
```

Lab Program 8

a) WAP: 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.

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

// Define a node structure for the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* 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 = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a node into the binary search tree
struct Node* insertNode(struct Node* root, int value) {
```

```
    if (root == NULL) {
    return createNode(value);
    }

    if (value < root->data) {
        root->left = insertNode(root->left, value);
    } else if (value > root->data) {
        root->right = insertNode(root->right, value);
    }

    return root;
}

// In-order traversal
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Pre-order traversal
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
```

```
}

// Post-order traversal
void postorderTraversal(struct Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

// Function to display the tree (in-order traversal as default)
void displayTree(struct Node* root) {
    if (root == NULL) {
        printf("Tree is empty.\n");
        return;
    }

    printf("Tree elements (In-order traversal): ");
    inorderTraversal(root);
    printf("\n");
}

int main() {
    struct Node* root = NULL;
    int choice, value;

    while (1) {
```

```
printf("\nBinary Search Tree Operations:\n");
printf("1. Insert Node\n");
printf("2. In-order Traversal\n");
printf("3. Pre-order Traversal\n");
printf("4. Post-order Traversal\n");
printf("5. Display Tree\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
  case 1:
    printf("Enter value to insert: ");
    scanf("%d", &value);
    root = insertNode(root, value);
    break;
  case 2:
    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n");
    break;
  case 3:
    printf("Pre-order Traversal: ");
    preorderTraversal(root);
    printf("\n");
    break;
  case 4:
    printf("Post-order Traversal: ");
```

```
            postorderTraversal(root);
            printf("\n");
            break;
        case 5:
            displayTree(root);
            break;
        case 6:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }
}
```

OUTPUT:

```
Menu:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter data: 23

Menu:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter data: 67

Menu:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter data: 45

Menu:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 2
In-order Traversal: 23 45 67

Menu:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 3
Pre-order Traversal: 23 67 45

Menu:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 4
Post-order Traversal: 45 67 23
```

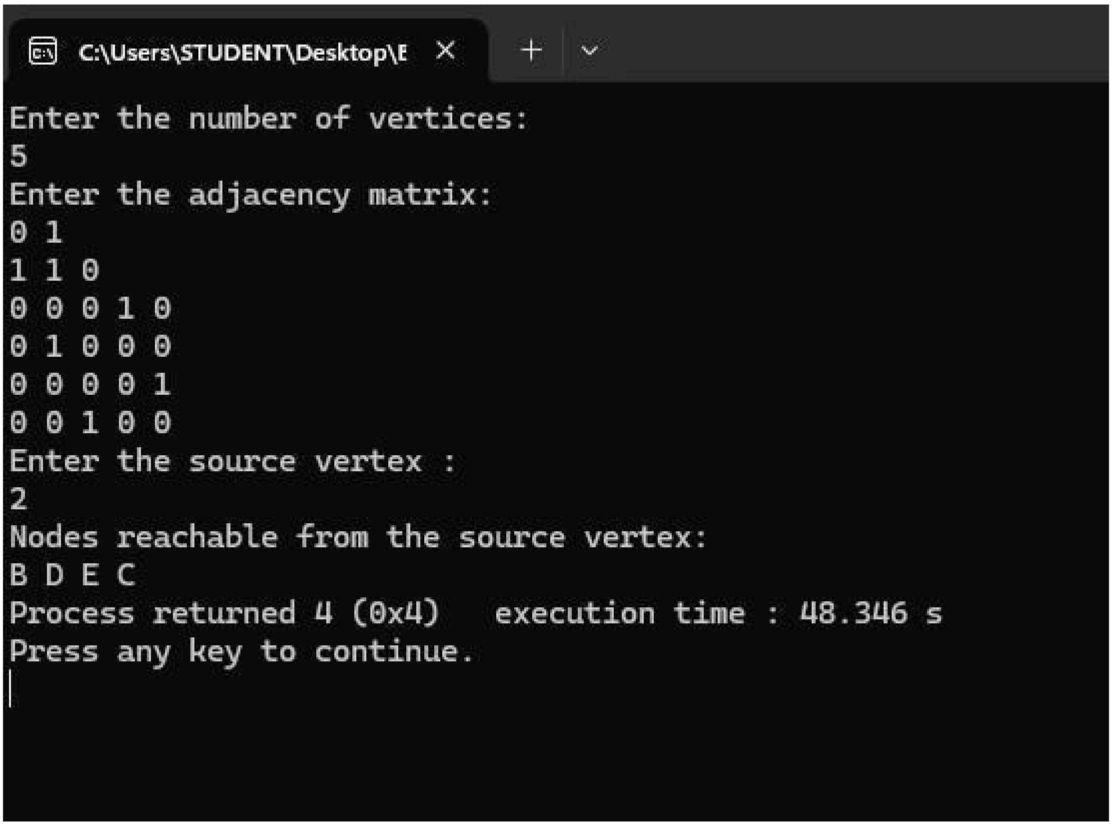
9a. Write a program to traverse a graph using BFS method.

```
#include <stdio.h>

#include

<stdlib.h>  #define

MAX 100

int

adj[MAX][MAX];

int visited[MAX];

int queue[MAX];

int front = -1, rear = -1;


void enqueue(int value) {

    if (rear == MAX - 1) {

        printf("Queue

        Overflow\n"); return;

    }

    if (front == -1) {

        front = 0;

    }

    queue[++rear] = value;

}


int dequeue() {

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

        printf("Queue Underflow\n");

        return -1;

    }

    return queue[front++];
```

```
}

void bfs(int start, int n) { for (int i = 0; i < n; i++) {
        visited[i] = 0;

    }


    enqueue(start);

    visited[start] = 1;


    printf("BFS Traversal: ");


    while (front <= rear) {

        int current = dequeue();

        printf("%d ", current);


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

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

                enqueue(i);

                visited[i] = 1;

            }

        }

    }

    printf("\n");

}


int main() {

    int n, edges, u, v, start;


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

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

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

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

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

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

    bfs(start, n);

    return 0;
}
```

Output:



```
C:\Users\STUDENT\Desktop\E
Enter the number of vertices:
5
Enter the adjacency matrix:
0 1
1 1 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
Enter the source vertex :
2
Nodes reachable from the source vertex:
B D E C
Process returned 4 (0x4)   execution time : 48.346 s
Press any key to continue.
```

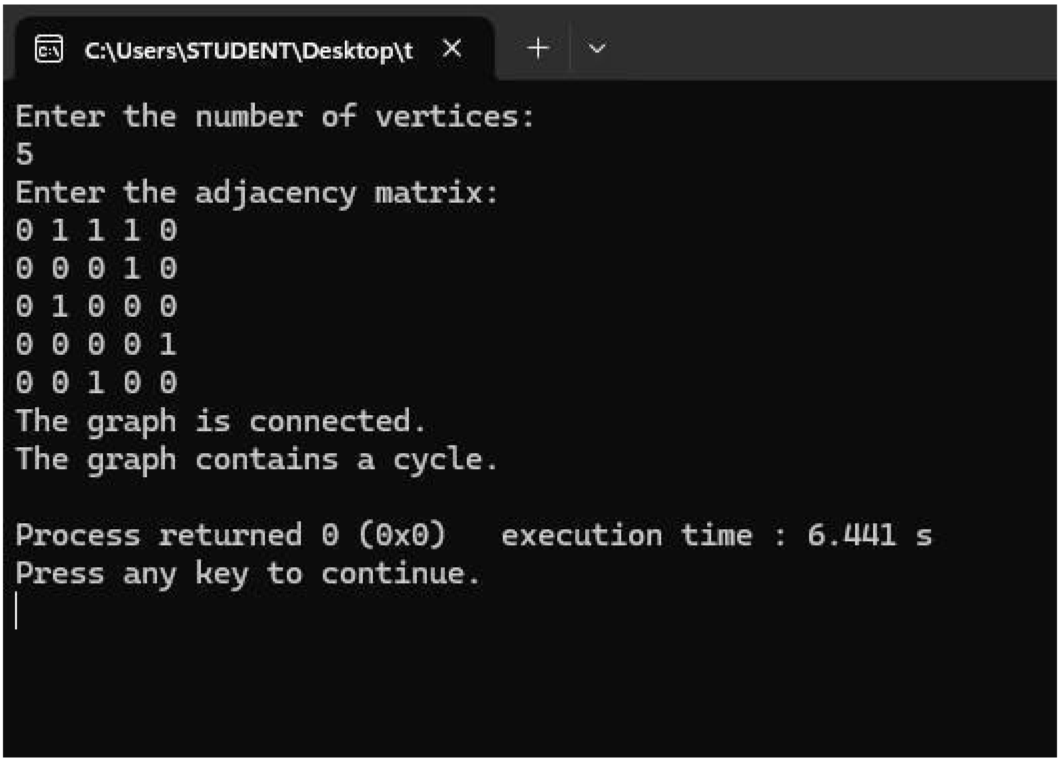
9b. Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
typedef struct Graph {
    int vertices;
    int adjMatrix[MAX_VERTICES][MAX_VERTICES];
    } Graph;
void initGraph(Graph* g, int vertices)
    { g->vertices = vertices;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            g->adjMatrix[i][j] = 0;
        }
    }
}
void addEdge(Graph* g, int u, int v)
    { g->adjMatrix[u][v] = 1;
    g->adjMatrix[v][u] = 1;
}
void DFS(Graph* g, int vertex, int visited[]) {
    visited[vertex] = 1;
    printf("%d ", vertex);
    for (int i = 0; i < g->vertices; i++) {
        if (g->adjMatrix[vertex][i] == 1 && !visited[i]) {
            DFS(g, i, visited);
```

```
        }
    }
}

int isConnected(Graph* g) {
    int visited[MAX_VERTICES] = {0};

    // Start DFS from the first vertex (0th index)
    DFS(g, 0, visited);
    for (int i = 0; i < g->vertices; i++) { if
    (visited[i] == 0) {
            return 0;
        }
    }
    return 1;
}

int main() {
    int vertices, edges, u, v;

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

    Graph g;
    initGraph(&g, vertices);

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

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

    printf("Enter edges (u v):\n");

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

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

        addEdge(&g, u, v);

    }

    if (isConnected(&g)) {

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

    } else {

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

    }

    return 0;

}
```

Output:



Lab Program 10

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

Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.

Let the keys in K and addresses in L are integers.

Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L.

Resolve the collision (if any) using linear probing.

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

#define MAX_EMPLOYEES 100
#define MAX_KEYS 10

#define m 10

typedef struct HashTable {
  int *table;
} HashTable;

void initHashTable(HashTable *ht) {
  ht->table = (int *)malloc(m * sizeof(int));
  for (int i = 0; i < m; i++) {
    ht->table[i] = -1;
  }
}

int hashFunction(int key) {
  return key % m;
}

void insert(HashTable *ht, int key) {
  int index = hashFunction(key);

  while (ht->table[index] != -1) {
    printf("Collision detected at index %d for key %d. Probing...\n", index, key);
    index = (index + 1) % m;
  }
```

```
    ht->table[index] = key;
    printf("Key %d inserted at index %d\n", key, index);
}
int search(HashTable *ht, int key) {
    int index = hashFunction(key);
    int originalIndex = index;


    while (ht->table[index] != -1) {
        if (ht->table[index] == key) {
            return index;
        }
        index = (index + 1) % m;
        if (index == originalIndex) {
            break;
        }
    }
    return -1;
}

void display(HashTable *ht) {
    printf("Hash Table Contents:\n");
    for (int i = 0; i < m; i++) {
        if (ht->table[i] != -1) {
            printf("Index %d: Key %d\n", i, ht->table[i]);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}

int main() {
    HashTable ht;
    initHashTable(&ht)
    ;

    int keys[MAX_KEYS], n, key;

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

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


    for (int i = 0; i < n; i++) {
        insert(&ht, keys[i]);
```

```
    }
    display(&ht);

    printf("Enter a key to search in the hash table: ");

    scanf("%d", &key);

    int index = search(&ht, key);
    if (index != -1) {
        printf("Key %d found at index %d\n", key, index);
    } else {
        printf("Key %d not found in the hash table\n", key);
    }


    free(ht.table);

    return 0;
}
```

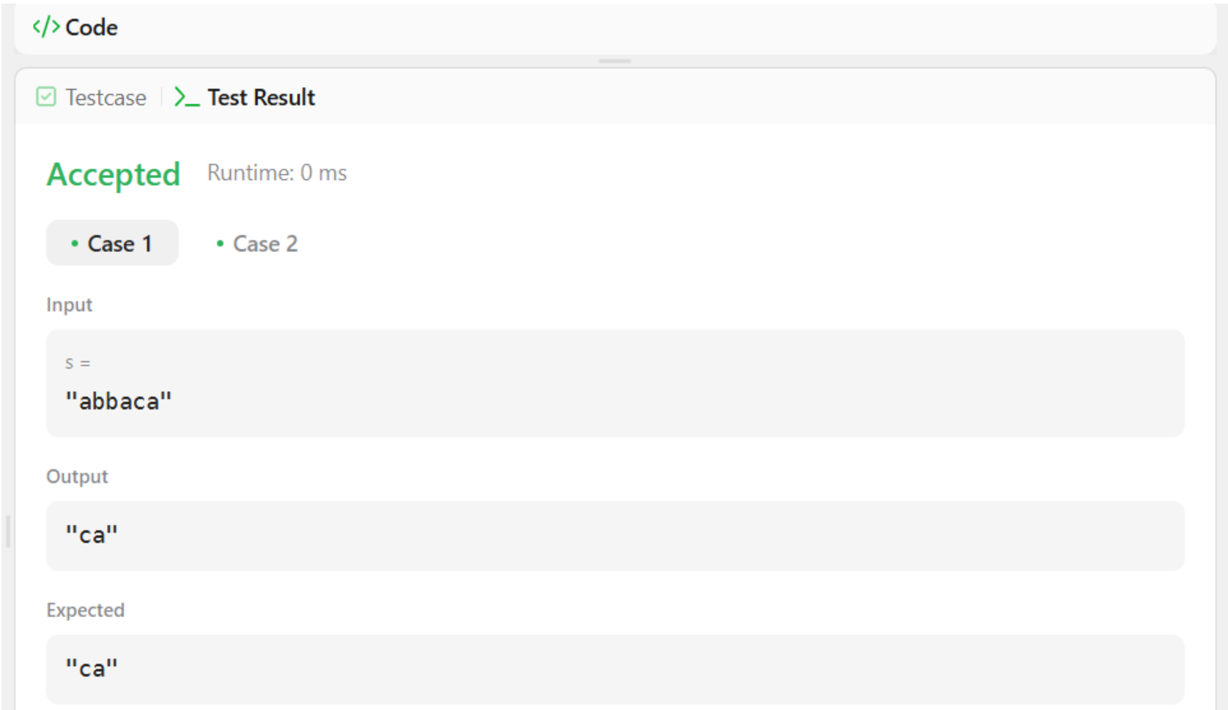
Output:

```
Enter the number of employee  records (N) :   5

Enter the two digit memory locations (m) for hash table:   11

Enter the four digit key values (K) for N Employee Records:
  1234
1245
1678
1908
3456

Hash Table contents are:

 T[0] --> -1
 T[1] --> -1
 T[2] --> 1234
 T[3] --> 1245
 T[4] --> 3456
 T[5] --> 1908
 T[6] --> 1678
 T[7] --> -1
 T[8] --> -1
 T[9] --> -1
 T[10] --> -1
```

LEETCODE PROBLEM – 1

```
char* removeDuplicates(char* s) {
    int n = strlen(s);
    char* stack = (char*)malloc(sizeof(char) * (n + 1));
    if (!stack) return NULL;
    int i = 0;
    for (int j = 0; j < n; j++) {
        char c = s[j];
        if (i && stack[i - 1] == c)
            { i--;
        } else {

            stack[i++] = c;
        }

    }
    stack[i] = '\0';
    return stack;
}
```

OUTPUT:



LEETCODE PROBLEM – 2

```
void backspace(const char* s, char* result) {
    int index = 0;
    for (int i = 0; s[i] != '\0'; i++) {
        if (s[i] != '#') {
            result[index++] = s[i];
        } else if (index > 0) {
            index--;
        }
    }
    result[index] = '\0';
}

bool backspaceCompare(char* s, char* t) {
    char resultS[1000];
    char resultT[1000];

    backspace(s,
    resultS); backspace(t,
    resultT);

    return strcmp(resultS, resultT) == 0;
}
```

OUTPUT:

Code

Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =

```
"ab#c"
```

t =

```
"ad#c"
```

Output

```
true
```

Expected

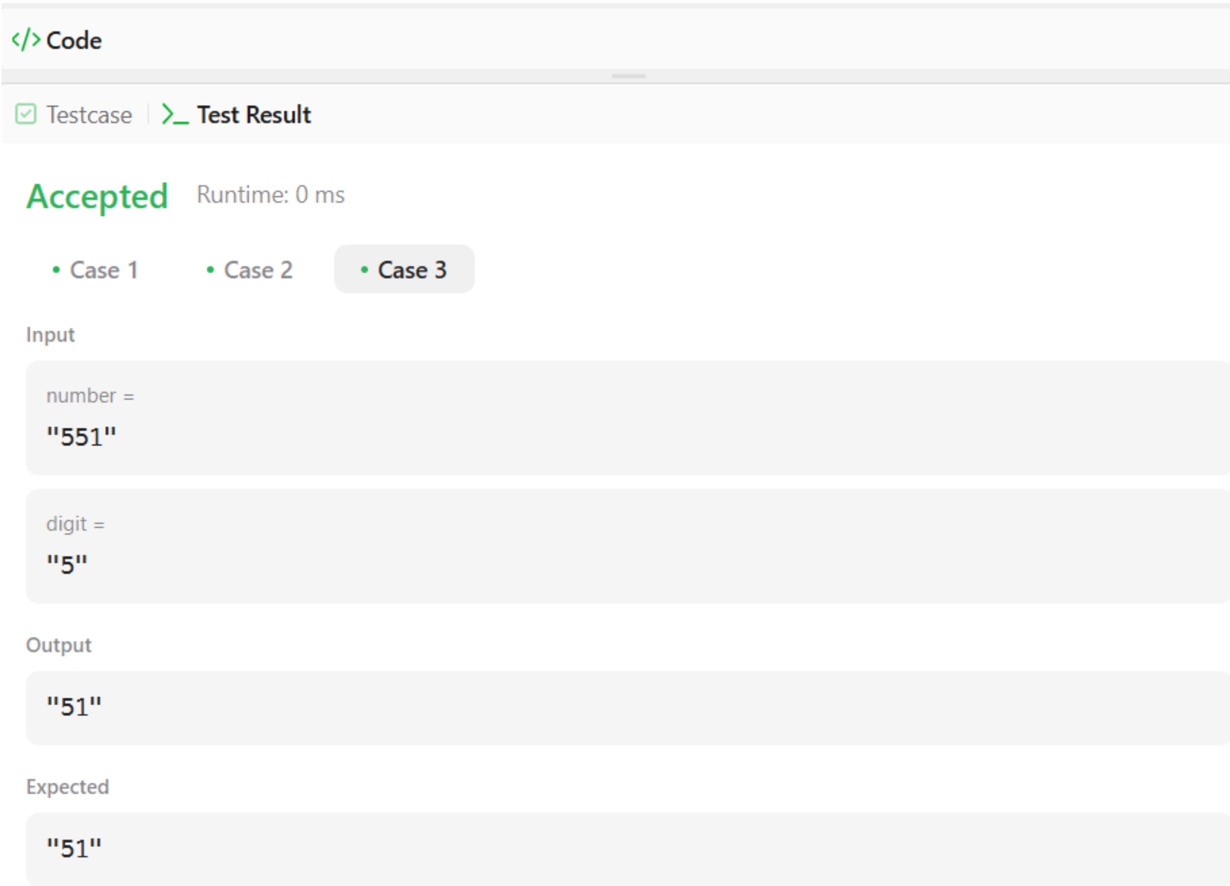
```
true
```

LEETCODE PROGRAM – 3

```
char* removeDigit(char* number, char digit){
	int n = strlen(number);
	static char result[100];
	int maxIndex = -1;

	for(i=0;i<n;i++){
		if(number[i]==digit){
			char temp[100];
			int k = 0;
			for(j=0;j<n;j++)
			{
				if(j!=i){
					temp[k++]=number[j];
				}
			}
			temp[k] = '\0';
			if(maxIndex==-1 || strcmp(temp,result)>0){
				strcpy(result,temp);
				maxIndex=I;
		}
	}
}
```

```
  return result;
}
```

OUTPUT:



LEETCODE PROBLEM 4

```
struct ListNode* deleteDuplicates(struct ListNode* head)
{

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

    struct ListNode* current = head;


    while (current != NULL && current->next != NULL) {

        if (current->val == current->next->val) {

            struct ListNode* temp =
            current->next; current->next =
            current->next->next; free(temp);
        } else {

            current = current->next;
        }
    }

    return head;
}
```

OUTPUT:

</> Code

Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

```
[1,1,2]
```

Output

```
[1,2]
```

Expected

```
[1,2]
```

LEETCODE PROBLEM 5

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

    if (head == NULL || head->next == 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;
}
```

OUTPUT:

Code

Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =

```
[3,2,0,-4]
```

pos =

```
1
```

Output

```
true
```

Expected

```
true
```