VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

MAHENDRA GOWDA (1BM24CS156)

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 MAHENDRA GOWDA **(1BM24CS156)**, 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 2025-26. 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

9a	Write a program to traverse a graph using BFS method.	45
9b	Write a program to check whether given graph is connected or not using DFS method.	47
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.	50
11	Leet code program **141. Linked List Cycle**	54
12	Leet code program **19. Remove Nth Node From End of List**	56
13	Leet code program **234. Palindrome Linked List**	58
14	Leet code program **1971. Find if Path Exists in Graph**	60

Course outcomes:

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

Lab program 1:

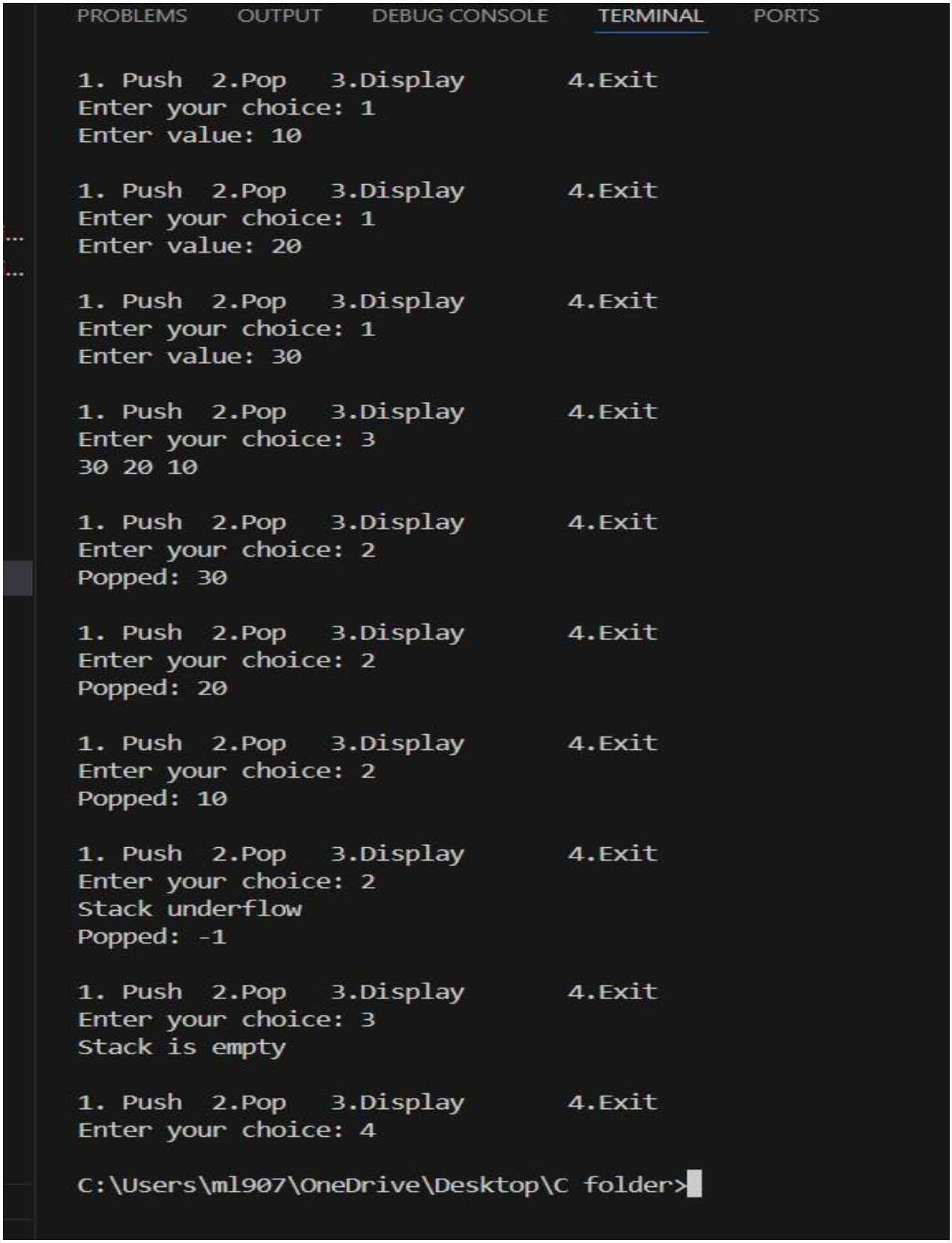
Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

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

#define MAX 100

typedef struct {
int arr[MAX];
int top;
} Stack;

void init(Stack *s) {     s-
>top = -1;
}

int isEmpty(Stack *s) {
return s->top == -1;
}

int isFull(Stack *s) {
return s->top == MAX - 1;
}

void push(Stack *s, int value) {
if (isFull(s)) {
printf("Stack overflow\n");
        return;
    }
    s->arr[++s->top] = value;
}

int pop(Stack *s) {     if
(isEmpty(s)) {         printf("Stack
underflow\n");
```

```
        return -1;
    }
    return s->arr[s->top--];
}

int peek(Stack *s) {     if
(isEmpty(s)) {
printf("Stack is empty\n");
return -1;
    }
    return s->arr[s->top];
}

int main(void) {
Stack s;     init(&s);

    int choice, value;

    while (1) {
printf("\n1. Push\n");
printf("2. Pop\n");
printf("3. Display\n");
printf("4. Exit\n");
printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
case 1:
                printf("Enter value: ");
scanf("%d", &value);
push(&s, value);                break;
            case
2:
                printf("Popped: %d\n", pop(&s));
break;
            case
3:
                if (isEmpty(&s)) {
printf("Stack is empty\n");
                } else {                    for (int i =
s.top; i >= 0; i--) {
printf("%d ", s.arr[i]);
                    }
printf("\n");                }
break;
            case 4:
exit(0);
```

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

    return 0;
}
```

Output:



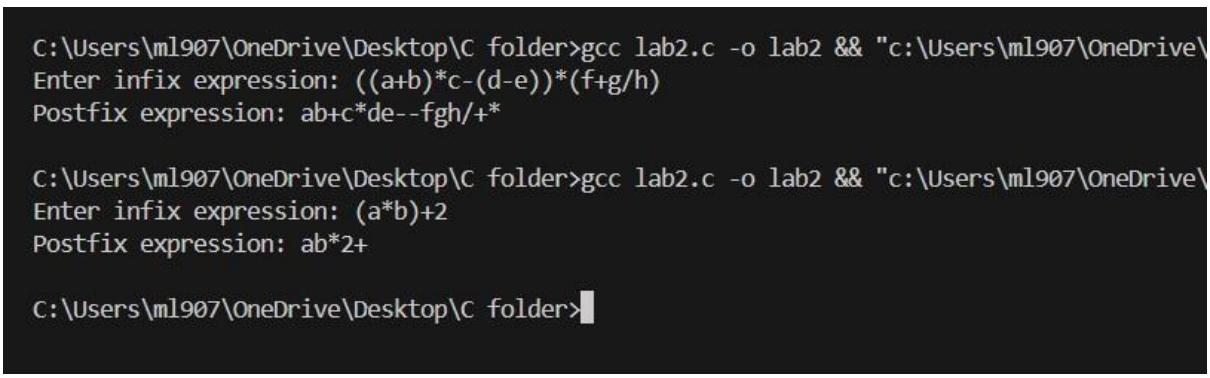
Lab Program 2:

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--];
}

char peek() {
   return stack[top];
}

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

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

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

   for (i = 0; infix[i] != '\0'; i++) {
char ch = infix[i];

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

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

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

    postfix[k] = '\0';

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

    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>
#include <stdlib.h>

#define MAX 3

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

void insert()
{
   int value;

   if (rear == MAX - 1)
   {
      printf("Queue overflow\n");
return;
   }

   printf("Enter value to insert: ");
   scanf("%d", &value);

   if (front == -1)
      front = 0;

   queue[++rear] = value;
}

void delete()
{
   if (front == -1 || front > rear)
   {
      printf("Queue is empty\n");
return;
   }

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

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

   printf("Queue elements: ");
   for (int i = front; i <= rear; i++)
   {
      printf("%d ", queue[i]);
   }
   printf("\n");
}

int main()
{    int
choice;

   while (1)
   {
      printf("\n1. Insert\t 2.Delete\t 3.Display\t 4.Exit\n");
printf("Enter choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
      case 1:
insert();
         break;

      case 2:
delete();
         break;

      case 3:
display();
         break;

      case 4:
exit(0);
```

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

    return 0;
}
```

Output:

b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The

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

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

#define MAX 3

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

void insert()
{
   int value;

   if ((rear + 1) % MAX == front)
   {
      printf("Queue overflow\n");
return;
   }

   printf("Enter value to insert: ");
   scanf("%d", &value);

   if (front == -1)
   {
      front = rear = 0;
   }
   else
   {
      rear = (rear + 1) % MAX;
   }

   queue[rear] = value;
}

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

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

    if (front == rear)
    {
        front = rear = -1;
    }
    else
    {
        front = (front + 1) % MAX;
    }
}

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

    printf("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("\n1. Insert\t 2.Delete\t 3.Display\t 4.Exit\n");
printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
```

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

        case 2:
delete();
            break;

        case 3:
display();
            break;

        case 4:
exit(0);

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

    return 0;
}
```

Output:

```
C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab3b.c -o lab3b && "c:\Us

1. Insert       2.Delete       3.Display      4.Exit
Enter choice: 1
Enter value to insert: 50

1. Insert       2.Delete       3.Display      4.Exit
Enter choice: 1
Enter value to insert: 60

1. Insert       2.Delete       3.Display      4.Exit
Enter choice: 1
Enter value to insert: 70

1. Insert       2.Delete       3.Display      4.Exit
Enter choice: 1
Queue overflow

1. Insert       2.Delete       3.Display      4.Exit
Enter choice: 3
Queue elements: 50 60 70

1. Insert       2.Delete       3.Display      4.Exit
Enter choice: 2
Deleted element: 50

1. Insert       2.Delete       3.Display      4.Exit
Enter choice: 2
Deleted element: 60

1. Insert       2.Delete       3.Display      4.Exit
Enter choice: 2
Deleted element: 70

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

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

C:\Users\ml907\OneDrive\Desktop\C folder>
```

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;

void create()
{
   int n, value;
   struct Node *temp, *newNode;

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

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

      newNode->data = value;
      newNode->next = NULL;

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

```
void insertAtBeginning()
{
   struct Node *newNode;
   int value;

   newNode = (struct Node *)malloc(sizeof(struct Node));
   printf("Enter data: ");
   scanf("%d", &value);

   newNode->data = value;     newNode-
>next = head;
   head = newNode;
}

void insertAtEnd()
{
   struct Node *newNode, *temp;
   int value;

   newNode = (struct Node *)malloc(sizeof(struct Node));
   printf("Enter data: ");
   scanf("%d", &value);

   newNode->data = value;
   newNode->next = NULL;

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

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

   temp->next = newNode;
}

void insertAtPosition()
{
   struct Node *newNode, *temp;
```

```
    int value, pos, i;

    printf("Enter position: ");
scanf("%d", &pos);
    if (pos == 1)
    {
        insertAtBeginning();
        return;
    }

    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("Enter data: ");
    scanf("%d", &value);

    newNode->data = value;

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

    printf("Linked list: ");
```

```
    while (temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main()
{
    int choice;

    while (1)
    {
        printf("\n1. Create List\t 2. Insert at Beginning\t 3.Insert at Position\t 4. Insert at End\t
5. Display\t 6. Exit\n");
printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
create();
            break;

        case 2:
insertAtBeginning();
            break;

        case 3:
insertAtPosition();
            break;

        case 4:
insertAtEnd();
            break;

        case 5:
display();
            break;

        case 6:
exit(0);
```

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

    return 0;
}
```

Output:

```
C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab4.c -o lab4 && "c:\Users\ml907\OneDrive\Desktop\C folder\"\lab4

1. Create List   2. Insert at Beginning  3.Insert at Position    4. Insert at End       5. Display     6. Exit
Enter choice: 1
Enter number of nodes: 4
Enter data: 10
Enter data: 20
Enter data: 30
Enter data: 40

1. Create List   2. Insert at Beginning  3.Insert at Position    4. Insert at End       5. Display     6. Exit
Enter choice: 5
Linked list: 10 -> 20 -> 30 -> 40 -> NULL

1. Create List   2. Insert at Beginning  3.Insert at Position    4. Insert at End       5. Display     6. Exit
Enter choice: 2
Enter data: 50

1. Create List   2. Insert at Beginning  3.Insert at Position    4. Insert at End       5. Display     6. Exit
Enter choice: 4
Enter data: 60

1. Create List   2. Insert at Beginning  3.Insert at Position    4. Insert at End       5. Display     6. Exit
Enter choice: 5
Linked list: 50 -> 10 -> 20 -> 30 -> 40 -> 60 -> NULL

1. Create List   2. Insert at Beginning  3.Insert at Position    4. Insert at End       5. Display     6. Exit
Enter choice: 3
Enter position: 3
Enter data: 70

1. Create List   2. Insert at Beginning  3.Insert at Position    4. Insert at End       5. Display     6. Exit
Enter choice: 5
Linked list: 50 -> 10 -> 70 -> 20 -> 30 -> 40 -> 60 -> NULL

1. Create List   2. Insert at Beginning  3.Insert at Position    4. Insert at End       5. Display     6. Exit
Enter choice: 6

C:\Users\ml907\OneDrive\Desktop\C folder>
```

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;

void create()
{
   int n, value;
   struct Node *temp, *newNode;

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

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

      newNode->data = value;
      newNode->next = NULL;

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

```
void deleteFirst()
{
  struct Node *temp;
  if (head == NULL)
   {
      printf("List is empty\n");
      return;
   }

   temp = head;
head = head->next;
   free(temp);

   printf("First node deleted\n");
}

void deleteLast()
{
   struct Node *temp, *prev;

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

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

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

   prev->next = NULL;
   free(temp);

   printf("Last node deleted\n");
}
```

```
void deleteSpecified()
{
   int value;
  struct Node *temp, *prev;

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

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

   if (head->data == value)
   {
      deleteFirst();
      return;
   }

   prev = head;
temp = head->next;

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

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

   prev->next = temp->next;
   free(temp);

   printf("Specified node deleted\n");
}

void display()
{
   struct Node *temp = head;

   if (head == NULL)
   {
```

```
        printf("List is empty\n");
        return;
    }

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

int main()
{
    int choice;

    while (1)
    {
        printf("\n1. Create List\t 2. Delete First\t 3. Delete Specified\t 4. Delete Last\t 5. Display\t 6.
Exit\n");        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
create();
            break;

        case 2:
deleteFirst();
break;

        case 3:
deleteSpecified();
            break;

        case 4:
deleteLast();
            break;

        case 5:
display();
            break;

        case 6:
exit(0);
```

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

    return 0;
}
```

Output:

```
C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab5.c -o lab5 && "c:\Users\ml907\OneDrive\Desktop\C folder\"\lab5

1. Create List   2. Delete First        3. Delete Specified     4. Delete Last  5. Display      6. Exit
Enter choice: 1
Enter number of nodes: 6
Enter data: 10
Enter data: 20
Enter data: 30
Enter data: 40
Enter data: 50
Enter data: 60

1. Create List   2. Delete First        3. Delete Specified     4. Delete Last  5. Display      6. Exit
Enter choice: 5
Linked list: 10 -> 20 -> 30 -> 40 -> 50 -> 60 -> NULL

1. Create List   2. Delete First        3. Delete Specified     4. Delete Last  5. Display      6. Exit
Enter choice: 2
First node deleted

1. Create List   2. Delete First        3. Delete Specified     4. Delete Last  5. Display      6. Exit
Enter choice: 4
Last node deleted

1. Create List   2. Delete First        3. Delete Specified     4. Delete Last  5. Display      6. Exit
Enter choice: 5
Linked list: 20 -> 30 -> 40 -> 50 -> NULL

1. Create List   2. Delete First        3. Delete Specified     4. Delete Last  5. Display      6. Exit
Enter choice: 3
Enter value to delete: 40
Specified node deleted

1. Create List   2. Delete First        3. Delete Specified     4. Delete Last  5. Display      6. Exit
Enter choice: 5
Linked list: 20 -> 30 -> 50 -> NULL

1. Create List   2. Delete First        3. Delete Specified     4. Delete Last  5. Display      6. Exit
Enter choice: 6

C:\Users\ml907\OneDrive\Desktop\C folder>
```

Lab program 6:

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

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

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

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

struct Node *createList()
{
   struct Node *head = NULL, *temp = NULL, *newNode;
int n, value;

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

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

      newNode->data = value;
      newNode->next = NULL;

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

```
    return head;
}

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

    while (head != NULL)
    {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

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

    for (i = head; i != 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, *curr = head, *next = NULL;

    while (curr != NULL)
    {
```

```
        next = curr->next;        curr-
>next = prev;       prev = curr;
curr = next;
   }
   return prev;
}

struct Node *concatenate(struct Node *head1, struct Node *head2)
{
   struct Node *temp;

   if (head1 == NULL)
      return head2;

   temp = head1;     while
(temp->next != NULL)
   {
      temp = temp->next;
   }

   temp->next = head2;
   return head1;
}

int main()
{
   printf("Create first linked list\n");
head1 = createList();

   printf("\nCreate second linked list\n");
head2 = createList();

   printf("\nFirst list:\n");
   display(head1);

   printf("Second list:\n");
   display(head2);

   sortList(head1);
printf("\nSorted first list:\n");
   display(head1);
```

```
    head1 = reverseList(head1);
  printf("\nReversed first list:\n");
  display(head1);

    head1 = concatenate(head1, head2);
    printf("\nConcatenated list:\n");
  display(head1);

    return 0;
  }
```

Output:

```
C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab6.c -o lab6 && "c:
Create first linked list
Enter number of nodes: 4
Enter data: 10
Enter data: 50
Enter data: 30
Enter data: 70

Create second linked list
Enter number of nodes: 5
Enter data: 90
Enter data: 67
Enter data: 34
Enter data: 60
Enter data: 20

First list:
10 -> 50 -> 30 -> 70 -> NULL
Second list:
90 -> 67 -> 34 -> 60 -> 20 -> NULL

Sorted first list:
10 -> 30 -> 50 -> 70 -> NULL

Reversed first list:
70 -> 50 -> 30 -> 10 -> NULL

Concatenated list:
70 -> 50 -> 30 -> 10 -> 90 -> 67 -> 34 -> 60 -> 20 -> NULL

C:\Users\ml907\OneDrive\Desktop\C folder>
```

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;
struct Node *front = NULL;
struct Node *rear = NULL;
```

```
void push()
{
   struct Node *newNode;
   int value;

   newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter value: ");
   scanf("%d", &value);

   newNode->data = value;     newNode-
>next = top;
   top = newNode;
}

void pop()
{
   struct Node *temp;

   if (top == NULL)
   {
      printf("Stack is empty\n");
      return;
   }

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

void displayStack()
{
   struct Node *temp = top;

   if (top == NULL)
   {
      printf("Stack is empty\n");
      return;
   }

   printf("Stack: ");
   while (temp != NULL)
   {
```

```
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void enqueue()
{
    struct Node *newNode;
    int value;

    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("Enter value: ");
    scanf("%d", &value);

    newNode->data = value;
    newNode->next = NULL;

    if (front == NULL)
    {
        front = rear = newNode;
    }
    else
    {
        rear->next = newNode;
       rear = newNode;
    }
}

void dequeue()
{
    struct Node *temp;

    if (front == NULL)
    {
        printf("Queue is empty\n");
return;
    }

    temp = front;
    printf("Deleted element: %d\n", temp->data);
    front = front->next;
    free(temp);
```

```
    if (front == NULL)
        rear = NULL;
}

void displayQueue()
{
    struct Node *temp = front;

    if (front == NULL)
    {
        printf("Queue is empty\n");
return;
    }

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

int main()
{ int choice;
    while (1)
    {
        printf("\n1. Push (Stack)\t 2. Pop (Stack)\t 3. Display Stack\t 4. Enqueue (Queue)\t
5. Dequeue (Queue)\t 6. Display Queue\t 7. Exit\n");        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
push();
            break;

        case 2:
pop();
            break;

        case 3:
displayStack();
            break;
```

```
      case 4:
enqueue();
         break;

      case 5:
dequeue();
         break;

      case 6:
displayQueue();
         break;

      case 7:
exit(0);

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

   return 0;
}
```

Output:

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;
```

```
   struct Node *next;
};

struct Node *head = NULL;

void create()
{
   int n, value;
   struct Node *newNode, *temp = NULL;

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

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

      newNode->data = value;        newNode-
>prev = NULL;        newNode->next = NULL;

      if (head == NULL)
      {
         head = newNode;
      }
else
      {
         temp = head;
         while (temp->next != NULL)
            temp = temp->next;

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

void insertLeft()
{    int value,
key;
   struct Node *newNode, *temp;

   if (head == NULL)
```

```
   {
      printf("List is empty\n");
      return;
   }

   printf("Enter new value: ");
scanf("%d", &value);
   printf("Insert to the left of node having value: ");
scanf("%d", &key);

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

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

   newNode = (struct Node *)malloc(sizeof(struct Node));
newNode->data = value;

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

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

   temp->prev = newNode;
}

void deleteNode()
{
   int key;    struct
Node *temp;

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

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

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

   if (temp == NULL)
   {
      printf("Node 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\n");
}

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");
}
```

```
int main()
{
   int choice;

   while (1)
   {
      printf("\n1. Create List\t 2. Insert Left\t 3. Delete Node\t 4. Display\t 5. Exit\n");
printf("Enter choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
      case 1:
create();
         break;

      case 2:
insertLeft();
         break;

      case 3:
deleteNode();
break;

      case 4:
        display();
         break;

     case 5:
exit(0);

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

   return 0;
}
```

Output:

```
1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
Enter choice: 1
Enter number of nodes: 4
Enter data: 30
Enter data: 60
Enter data: 74
Enter data: 99

1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
Enter choice: 2
Enter new value: 45
Insert to the left of node having value: 74

1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
Enter choice: 4
Doubly linked list: 30 <-> 60 <-> 45 <-> 74 <-> 99 <-> NULL

1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
Enter choice: 2
Enter new value: 68
Insert to the left of node having value: 77
Specified node not found

1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
Enter choice: 3
Enter value to delete: 44
Node not found

1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
Enter choice: 45
Invalid choice

1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
Enter choice: 3
Enter value to delete: 45
Node deleted

1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
Enter choice: 4
Doubly linked list: 30 <-> 60 <-> 74 <-> 99 <-> NULL

1. Create List   2. Insert Left  3. Delete Node  4. Display      5. Exit
```

Lab program 8:

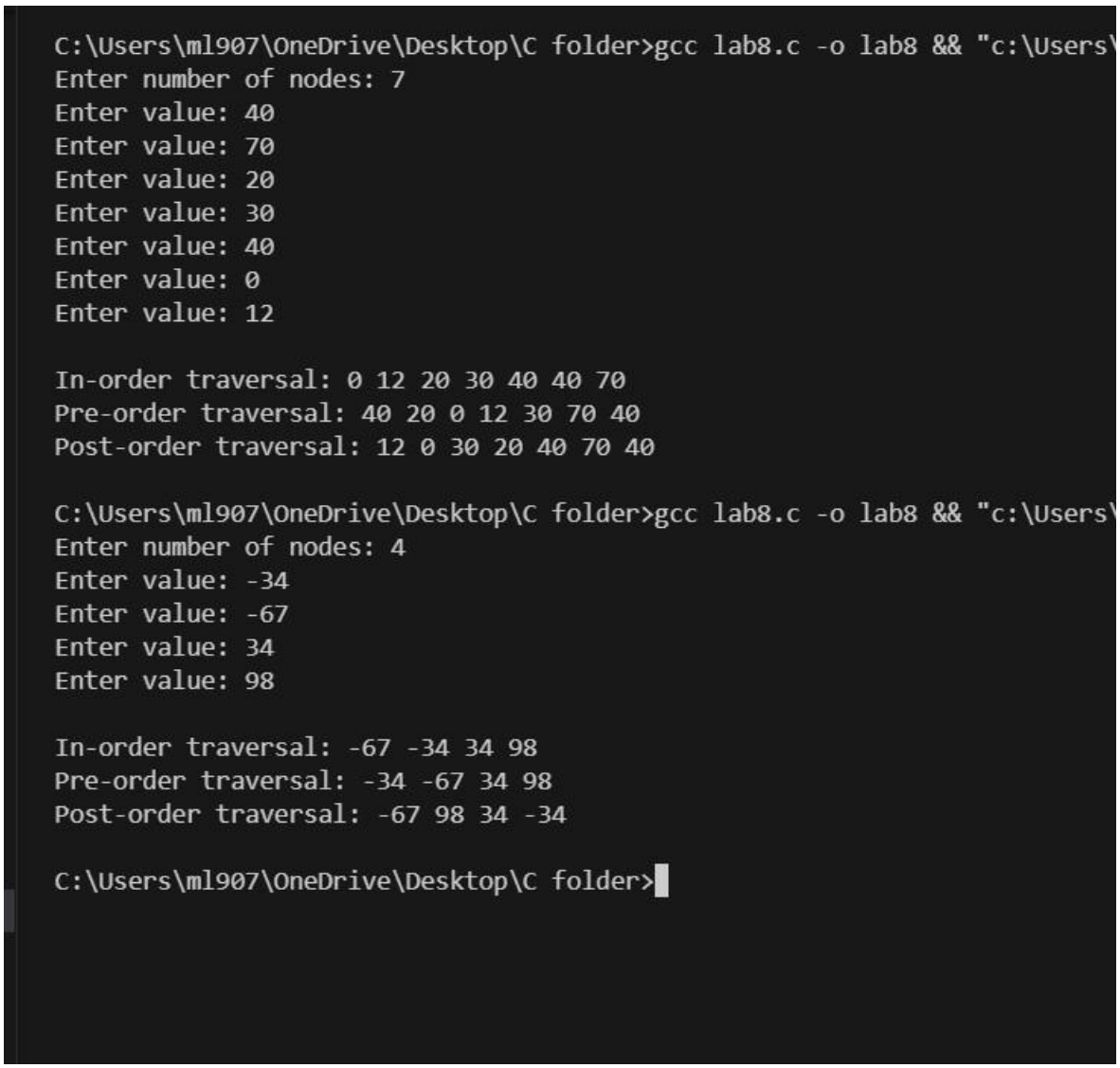
Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree.

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

```
struct Node
{    int data;
struct Node *left;
   struct Node *right;
};

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;
}

struct Node *insert(struct Node *root, int value)
{
   if (root == NULL)
      return createNode(value);

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

   return root;
}

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

void preorder(struct Node *root)
{
   if (root != NULL)
   {
      printf("%d ", root->data);       preorder(root-
>left);
```

```
        preorder(root->right);
    }
}

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 n, value;

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

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

    printf("\nIn-order traversal: ");
    inorder(root);

    printf("\nPre-order traversal: ");     preorder(root);

    printf("\nPost-order traversal: ");
postorder(root);

    printf("\n");

    return 0;
}
```

Output:



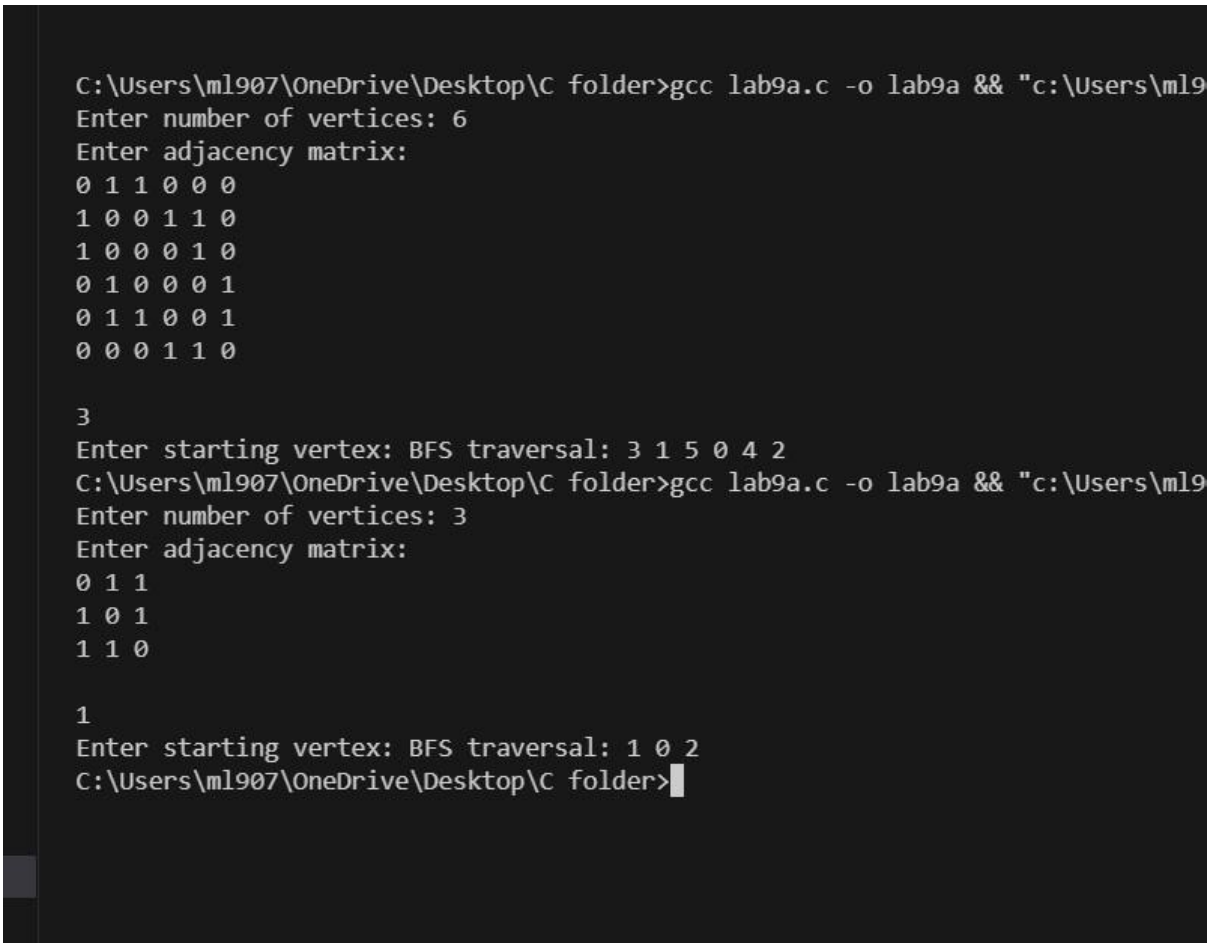
Lab Program 9:

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

```
#include <stdio.h>

#define MAX 10

int adj[MAX][MAX];
int visited[MAX];
int queue[MAX];
int front = 0, rear = -1;
int n;
```

```
void bfs(int start) {
   int i, v;

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

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

      for (i = 0; i < n; i++) {
         if (adj[v][i] == 1 && visited[i] == 0) {
queue[++rear] = i;
            visited[i] = 1;
         }
      }
   }
}

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", &adj[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;
}
```

Output:



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] == 0) {
dfs(i);
```

```
        }
    }
}

int main() {
    int i, j;
    int connected = 1;

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

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

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

    dfs(0);

    for (i = 0; i < n; i++) {
if (visited[i] == 0) {
connected = 0;
break;
        }
    }

    if (connected)         printf("Graph is
CONNECTED\n");
    else
        printf("Graph is NOT CONNECTED\n");

    return 0;
}
```

Output:

```
C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab9b.c -o lab9b && "c:\Users\ml907\OneDrive\Desktop\C folder\"\lab9b
Enter number of vertices: 4
Enter adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Graph is CONNECTED

C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab9b.c -o lab9b && "c:\Users\ml907\OneDrive\Desktop\C folder\"\lab9b
Enter number of vertices: 4
Enter adjacency matrix:
0 1 0 0
1 0 0 0
0 0 0 1
0 0 1 0
Graph is NOT CONNECTED

C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab9b.c -o lab9b && "c:\Users\ml907\OneDrive\Desktop\C folder\"\lab9b
Enter number of vertices: 5
Enter adjacency matrix:
0 1 1 0 0
1 0 0 1 1
1 0 0 0 1
0 1 0 0 0
0 1 1 0 0
Graph is CONNECTED

C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab9b.c -o lab9b && "c:\Users\ml907\OneDrive\Desktop\C folder\"\lab9b
Enter number of vertices: 3
Enter adjacency matrix:
0 1 1
1 0 1
1 1 0
Graph is CONNECTED

C:\Users\ml907\OneDrive\Desktop\C folder>
```

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;

void initialize()
{
   for (int i = 0; i < m; i++)
      hashTable[i] = -1;
}

void insert(int key)
{
   int index = key % m;
   int start = index;

   while (hashTable[index] != -1)
   {
      index = (index + 1) % m;
if (index == start)
      {
         printf("Hash table is full\n");
         return;
      }
   }

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

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);
    }
}

int main()
{    int n,
key;

    printf("Enter number of memory locations (m): ");
scanf("%d", &m);

    initialize();

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

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

    display();

    return 0;
}
```

Output:

```
C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab10.
Enter number of memory locations (m): 10
Enter number of keys: 6
Enter key: 1234
Key 1234 inserted at address 4
Enter key: 2356
Key 2356 inserted at address 6
Enter key: 2354
Key 2354 inserted at address 5
Enter key: 9867
Key 9867 inserted at address 7
Enter key: 3433
Key 3433 inserted at address 3
Enter key: 1111
Key 1111 inserted at address 1

Hash Table:
Address 0 : Empty
Address 1 : 1111
Address 2 : Empty
Address 3 : 3433
Address 4 : 1234
Address 5 : 2354
Address 6 : 2356
Address 7 : 9867
Address 8 : Empty
Address 9 : Empty
```

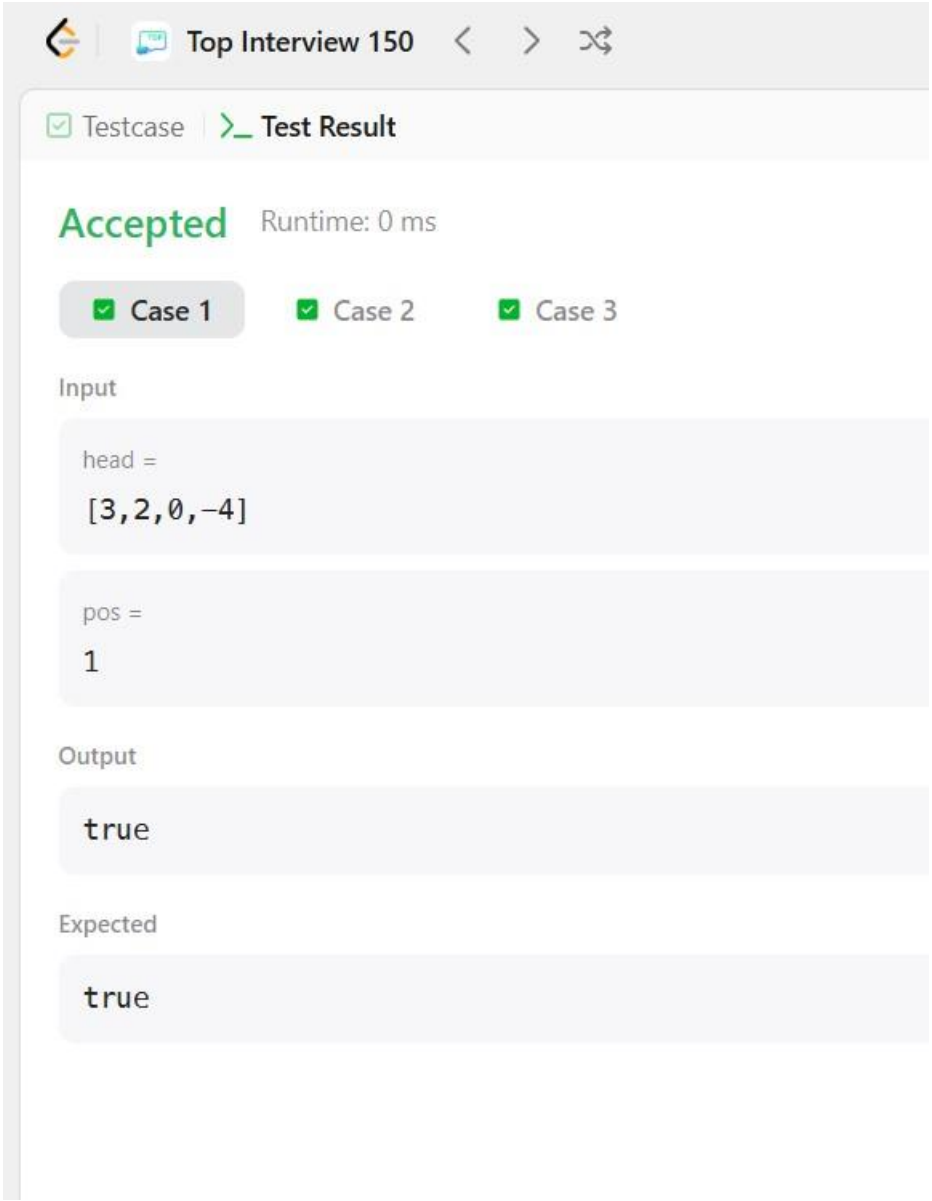
```
C:\Users\ml907\OneDrive\Desktop\C folder>gcc lab
Enter number of memory locations (m): 5
Enter number of keys: 6
Enter key: 2235
Key 2235 inserted at address 0
Enter key: 1234
Key 1234 inserted at address 4
Enter key: 5655
Key 5655 inserted at address 1
Enter key: 1986
Key 1986 inserted at address 2
Enter key: 2005
Key 2005 inserted at address 3
Enter key: 1888
Hash table is full

Hash Table:
Address 0 : 2235
Address 1 : 5655
Address 2 : 1986
Address 3 : 2005
Address 4 : 1234
```

Leet code programs: **141. Linked List Cycle**

```
bool hasCycle(struct ListNode *head) {
if(head==NULL || head->next==NULL)return false;
struct ListNode* slow = head;  struct ListNode* fast = head->next;  while(fast!=NULL && fast->next!=NULL){  slow = slow->next;  fast = fast->next->next;  if(slow==fast) return true;
 }
```

```
  return false;
}
```

Test cases:

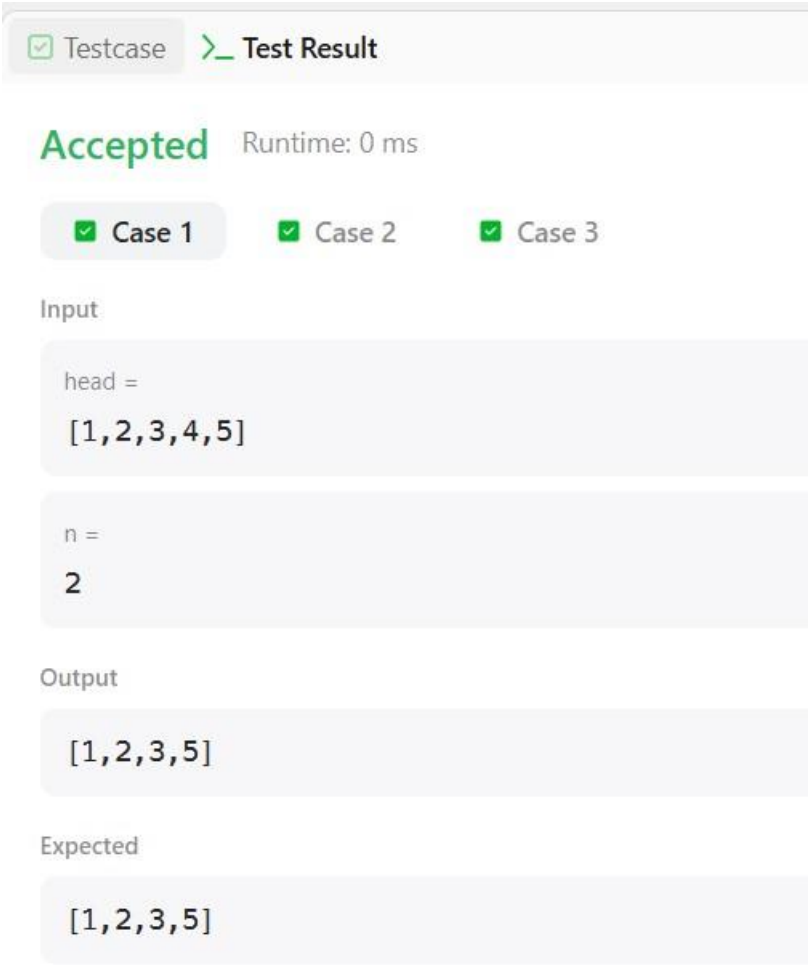


19. Remove Nth Node From End of List

```
struct ListNode* removeNthFromEnd(struct ListNode* head, int n) {
   int len = 0;     struct ListNode*
temp = head;     if(head-
>next==NULL) return NULL;
while(temp!=NULL){
      len++;       temp =
temp->next;
   }
```

```
   int x= len-n;     temp=head;
if(x==0)head = head->next;
for(int i=1;i<x;i++){        temp
= temp->next;
   }
   if(temp->next->next!=NULL)
temp->next = temp->next->next;
else temp->next=NULL;        return
head;
}
```

Test case:

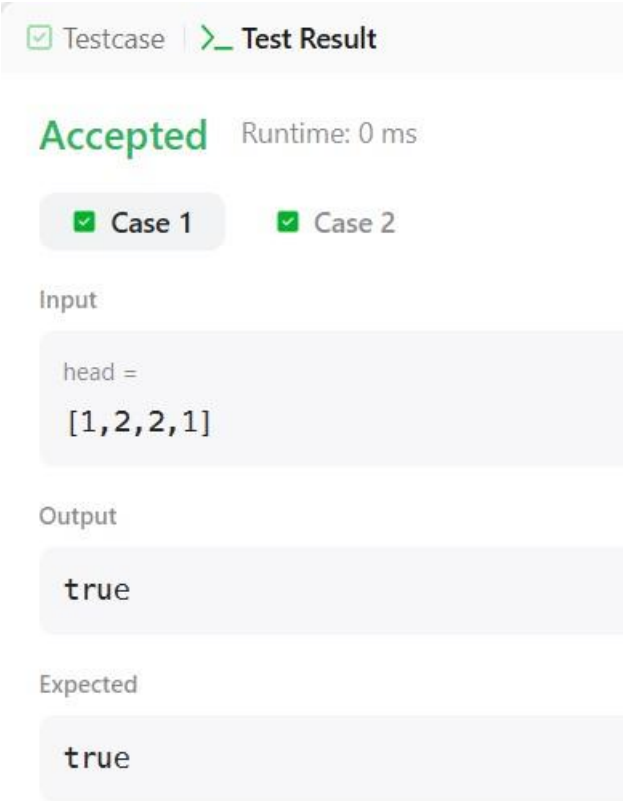


234. Palindrome Linked List

```
bool isPalindrome(struct ListNode* head) {
int arr[100000];    int n=0;    struct
ListNode* temp = head;
while(temp!=NULL){       arr[n++]=temp-
>val;       temp = temp->next;
  }
  int i=0, j=n-1;
while(i<j){
```

```
if(arr[i]!=arr[j]){
return false;
      }
i++;
j--;
   }
   return true;
}
```

Test case:



1971. Find if Path Exists in Graph

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

void dfs(int node, int **adj, int *adjSize, bool *visited) {
visited[node] = true;

   for (int i = 0; i < adjSize[node]; i++) {
int next = adj[node][i];        if
(!visited[next]) {
          dfs(next, adj, adjSize, visited);
      }
   }
}

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

   int **adj = (int **)malloc(n * sizeof(int *));
int *adjSize = (int *)calloc(n, sizeof(int));
   bool *visited = (bool *)calloc(n, sizeof(bool));

   for (int i = 0; i < edgesSize; i++) {
adjSize[edges[i][0]]++;
adjSize[edges[i][1]]++;
   }

   for (int i = 0; i < n; i++) {
      adj[i] = (int *)malloc(adjSize[i] * sizeof(int));
      adjSize[i] = 0;
   }

   for (int i = 0; i < edgesSize; i++) {
int u = edges[i][0];        int v =
edges[i][1];
adj[u][adjSize[u]++] = v;
      adj[v][adjSize[v]++] = u;
   }

   dfs(source, adj, adjSize, visited);

   return visited[destination];
}
```

Test case:


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
n =
3
edges =
[[0,1],[1,2],[2,0]]
source =
0
destination =
2
Output
true
Expected
true