Dayananda Sagar College of Engineering
Department of MCA
LABORATORY MANUAL

PROGRAM : MASTER OF COMPUTER APPLICATIONS

COURSE NAME : Problem Solving Technique Using C Laboratory

COURSE CODE : MMC106

SEMESTER : 1st SEMESTER

FACULTY : Prof.Alamma B H and Prof.Seema Aparaj

PART-A

1. **Write a C program using else-if ladder to print Grades of the student.**
2. **Write a C program using nesting for to print multiplication table 1-10.**
3. **Write a C program that uses functions to perform Addition, Subtraction, Multiplication, and Division of two Matrices.**
4. **Write a C program using recursion function and perform Tower of Hanoi**
5. **Write a C program to reverse the string using pointers**
6. **Write a C program to using array to calculate sum and average of the elements**
7. **Write a C program to implement linear search and binary search using switch case.**
8. **Write a C Program to implement Quick sort.**
9. **Write a C program to implement Circular Queue.**
10. **Write a C program to implement Doubly Linked List.**

PART-B

Case Study Question 1: E-Commerce Inventory & Order Processing System

Scenario:

You are asked to build the backend for a small e-commerce company that manages products and customer orders.

Tasks:

1. **Design structures to store product details and order details (include nested structures).**
2. **Write a C program to:**
   - **Add new products**
   - **Search for products**
   - **Update stock after customer order**
   - **Store completed orders in a binary file**
3. **Apply input validation and error handling.**
4. **Generate a report listing products with stock below a threshold (e.g., less than 10).**
5. **Discuss how dynamic memory allocation can improve performance in this system.**

Case Study Question 2: Banking Transaction Simulator

Scenario:

Simulate a banking system that supports account creation, deposit/withdrawal, and mini-statements.

Tasks:

1. Define a structure for bank accounts and transactions.
2. Write a C program for:
   - New account registration
   - Deposit and withdrawal
   - Validating balance during withdrawal
   - Saving transactional history using linked lists
3. Implement dynamic memory allocation for transaction nodes.
4. Generate a mini-statement showing last 5 transactions.

Case Study Question 3: Airline Reservation System

Scenario:

Develop an airline seat booking platform.

Tasks:

1. Represent seats using 2D arrays (rows & columns).
2. Write C code for:
    - Viewing seat availability
    - Booking and cancelling seats
    - Assigning seat numbers automatically
3. Store booking details using structures and binary files.
4. Generate report of all booked seats.

   Suggest improvements using dynamic memory.

Case Study Question 4: Smart Parking System

Scenario:

A mall wants an automated parking management system.

Tasks:

1. Use structures to store vehicle details and entry/exit times.
2. Implement:
    - Add vehicle
    - Remove vehicle
    - Calculate parking fee
3. Use file handling to maintain daily logs.
4. Add error checking (slot full, duplicate number).

PART- A

1. **Write a C program using else-if ladder to print Grades of the student**

```
#include <stdio.h>
  int main() {
    int marks;

    // Input marks from the user
    printf("Enter your marks (0-100): ");
    scanf("%d", &marks);

    // Check grade using if-else if ladder
    if (marks >= 90) {
        printf("Grade: A\n");
    }
    else if (marks >= 80) {
        printf("Grade: B\n");
    }
    else if (marks >= 70) {
        printf("Grade: C\n");
    }
    else if (marks >= 60) {
        printf("Grade: D\n");
    }
    else {
        printf("Grade: F\n");
    }
return 0;
}
```

2. **Write a C program using nesting for to print multiplication table 1-10.**

```
#include <stdio.h>

int main() {
    // Outer loop for rows (1 to 10)
    for (int i = 1; i <= 10; i++) {
        // Inner loop for columns (1 to 10)
        for (int j = 1; j <= 10; j++) {
            printf("%d\t", i * j);  // Print the multiplication result
        }
        printf("\n");  // Move to the next line after each row
    }

    return 0;
    }
```

3.Write a C program that uses functions to perform Addition, Subtraction, Multiplication, and Division of two Matrices.

```
#include <stdio.h>

#define MAX 10

void inputMatrix(int matrix[MAX][MAX], int row, int col) {
    printf("Enter the elements of the matrix:\n");
    for(int i = 0; i < row; i++) {
        for(int j = 0; j < col; j++) {
            printf("Enter element [%d][%d]: ", i+1, j+1);
            scanf("%d", &matrix[i][j]);
        }
    }
```

```
}

void displayMatrix(int matrix[MAX][MAX], int row, int col) {

    printf("The matrix is:\n");

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

        for(int j = 0; j < col; j++) {

            printf("%d ", matrix[i][j]);

        }

        printf("\n");

    }

}


void addMatrices(int matrix1[MAX][MAX], int matrix2[MAX][MAX], int
result[MAX][MAX], int row, int col) {

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

        for(int j = 0; j < col; j++) {
```

```
            result[i][j] = matrix1[i][j] + matrix2[i][j];

        }

    }

}


void subtractMatrices(int matrix1[MAX][MAX], int matrix2[MAX][MAX], int
result[MAX][MAX], int row, int col) {

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

        for(int j = 0; j < col; j++) {

            result[i][j] = matrix1[i][j] - matrix2[i][j];

        }

    }

}


void multiplyMatrices(int matrix1[MAX][MAX], int matrix2[MAX][MAX], int
result[MAX][MAX], int row1, int col1, int row2, int col2) {
```

```
    if (col1 != row2) {
        printf("Matrix multiplication is not possible.\n");
        return;
    }
    for(int i = 0; i < row1; i++) {
        for(int j = 0; j < col2; j++) {
            result[i][j] = 0;
            for(int k = 0; k < col1; k++) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
}

int main() {
```

```
int choice, row1, col1, row2, col2;

int matrix1[MAX][MAX], matrix2[MAX][MAX], result[MAX][MAX];

printf("Enter number of rows and columns for Matrix 1: ");

scanf("%d %d", &row1, &col1);

inputMatrix(matrix1, row1, col1);

printf("Enter number of rows and columns for Matrix 2: ");

scanf("%d %d", &row2, &col2);

inputMatrix(matrix2, row2, col2);


do {

    printf("\nMatrix Operations:\n");

    printf("1. Matrix Addition\n");

    printf("2. Matrix Subtraction\n");
```

```
printf("3. Matrix Multiplication\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

   case 1:

      if(row1 == row2 && col1 == col2) {

         addMatrices(matrix1, matrix2, result, row1, col1);

         printf("Result of Matrix Addition:\n");

         displayMatrix(result, row1, col1);

      } else {

          printf("Matrix dimensions must be the same for addition.\n");

      }

      break;
```

```
case 2:
    if(row1 == row2 && col1 == col2) {
        subtractMatrices(matrix1, matrix2, result, row1, col1);
        printf("Result of Matrix Subtraction:\n");
        displayMatrix(result, row1, col1);
    } else {
        printf("Matrix dimensions must be the same for subtraction.\n");
    }
    break;
case 3:
    multiplyMatrices(matrix1, matrix2, result, row1, col1, row2, col2);
    if (col1 == row2) {
        printf("Result of Matrix Multiplication:\n");
        displayMatrix(result, row1, col2);
    }
```

```
            break;

        case 4:

            printf("Exiting the program.\n");

            break;

        default:

            printf("Invalid choice! Please try again.\n");

    }

} while (choice != 4);



return 0;

}
```

4.Write a C program using recursion function and perform Tower of Hanoi

```
#include <stdio.h>

// Recursive function to solve Tower of Hanoi
void towerOfHanoi(int n, char fromRod, char toRod, char auxRod) {
    if (n == 1) {
        // Base case: Move one disk directly
        printf("Move disk 1 from %c to %c\n", fromRod, toRod);
        return;
    }
```

```
    // Move top n-1 disks from 'fromRod' to 'auxRod'
    towerOfHanoi(n - 1, fromRod, auxRod, toRod);

    // Move the nth disk from 'fromRod' to 'toRod'
    printf("Move disk %d from %c to %c\n", n, fromRod, toRod);

    // Move the n-1 disks from 'auxRod' to 'toRod'
    towerOfHanoi(n - 1, auxRod, toRod, fromRod);
}

int main() {
    int numDisks;

    // Input: Number of disks
    printf("Enter the number of disks: ");
    scanf("%d", &numDisks);

    // Solve Tower of Hanoi
    printf("The sequence of moves to solve Tower of Hanoi is:\n");
    towerOfHanoi(numDisks, 'A', 'C', 'B'); // 'A', 'C', 'B' are the rod names
```

```
    return 0;
}
```

5.Write a C program to reverse the string using pointers.

```
Reverse  the  string
#include   <stdio.h>
#include <string.h>

void reverseString(char* str) {
    char* start = str;
    char* end = str + strlen(str) - 1; // Pointer to the last character
    char temp;

    while (start < end) {
        temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);

    reverseString(str);

    printf("Reversed string: %s\n", str);
```

6.Write a C program to using array to calculate sum and average of the elements.

```
#include <stdio.h>

int main() {
    int n, i;
    float sum = 0, average;

    // Input: Number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Declare the array
    int arr[n];

    // Input: Elements of the array
    printf("Enter the elements: \n");
    for (i = 0; i < n; i++) {
       scanf("%d", &arr[i]);}
```

```
// Calculate the sum of elements
for (i = 0; i < n; i++) {
    sum += arr[i];
}

// Calculate the average
average = sum / n;

// Output the results
printf("Sum of elements: %.2f\n", sum);
printf("Average of elements: %.2f\n", average);

return 0;
}
```

7.Write a C program to implement linear search and binary search using switch case.

```
#include <stdio.h>

void linearSearch(int arr[], int n, int target) {
```

```
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            printf("Element %d found at index %d\n", target, i);
            return;
        }
    }
    printf("Element %d not found\n", target);
}

void binarySearch(int arr[], int n, int target) {
    int low = 0, high = n - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == target) {
            printf("Element %d found at index %d\n", target, mid);
            return;
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
```

```
    printf("Element %d not found\n", target);
}

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

    // Input: Number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Declare the array
    int arr[n];

    // Input: Elements of the array
    printf("Enter the elements: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Menu
    printf("Choose search method:\n");
    printf("1. Linear Search\n");
```

```
printf("2. Binary Search (Array must be sorted)\n");
scanf("%d", &choice);

// Input: Target element to search for
printf("Enter the element to search for: ");
scanf("%d", &target);

// Perform the chosen search
switch (choice) {
    case 1:
        linearSearch(arr, n, target);
        break;
    case 2:
        // First, we sort the array before binary search
        // Simple bubble sort for demonstration
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n-i-1; j++) {
                if (arr[j] > arr[j+1]) {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
```

```
            }
        }
        binarySearch(arr, n, target);
        break;
    default:
        printf("Invalid choice.\n");
}

return 0;
}
```

8.Write a C Program to implement Quick sort.

```
#include <stdio.h>

// Function to swap two elements
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

```
// Partition function for Quick Sort
int partition(int arr[], int low, int high) {
    int pivot = arr[high];  // choosing the last element as pivot
    int i = low - 1;  // index of smaller element

    for (int j = low; j <= high - 1; j++) {
        // If current element is smaller than or equal to the pivot
        if (arr[j] <= pivot) {
            i++;  // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);  // return the partition index
}

// Quick Sort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // partitioning index
        int pi = partition(arr, low, high);
```

```
        // Recursively sort the two subarrays
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int n;

    // Input: Number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Declare the array
```

```
    int arr[n];

    // Input: Elements of the array
    printf("Enter the elements: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Calling quickSort function
    quickSort(arr, 0, n - 1);

    // Output: Sorted array
    printf("Sorted array: \n");
    printArray(arr, n);

    return 0;
}
```

9.Write a C program to implement Circular Queue.

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

```
#define MAX 5  // Define the maximum size of the queue

// Circular Queue structure
typedef struct {
    int front, rear;
    int arr[MAX];
} CircularQueue;

// Function to initialize the queue
void initQueue(CircularQueue *q) {
    q->front = -1;
    q->rear = -1;
}

// Function to check if the queue is full
int isFull(CircularQueue *q) {
    return ((q->rear + 1) % MAX == q->front);
}

// Function to check if the queue is empty
int isEmpty(CircularQueue *q) {
    return (q->front == -1);
```

```
}

// Function to enqueue (insert) an element
void enqueue(CircularQueue *q, int value) {
    if (isFull(q)) {
        printf("Queue is full. Cannot enqueue %d\n", value);
    } else {
        if (q->front == -1) {
            q->front = 0;  // If the queue is empty, set front to 0
        }
        q->rear = (q->rear + 1) % MAX;  // Circular increment of rear
        q->arr[q->rear] = value;
        printf("%d enqueued to queue\n", value);
    }
}

// Function to dequeue (remove) an element
int dequeue(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Cannot dequeue\n");
        return -1;
    } else {
```

```
        int dequeuedValue = q->arr[q->front];
        if (q->front == q->rear) {
            q->front = q->rear = -1;  // Queue becomes empty
        } else {
            q->front = (q->front + 1) % MAX;  // Circular increment of front
        }
        return dequeuedValue;
    }
}

// Function to display the elements of the queue
void display(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
    } else {
        printf("Queue elements: ");
        int i = q->front;
        while (i != q->rear) {
            printf("%d ", q->arr[i]);
            i = (i + 1) % MAX;
        }
        printf("%d\n", q->arr[q->rear]);  // Print the last element
```

```
    }
}

int main() {
    CircularQueue q;
    initQueue(&q);

    // Test the Circular Queue operations
    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    enqueue(&q, 40);
    enqueue(&q, 50);
    enqueue(&q, 60);  // This will show "Queue is full"

    display(&q);

    printf("Dequeued: %d\n", dequeue(&q));
    printf("Dequeued: %d\n", dequeue(&q));

    enqueue(&q, 60);  // Now the queue has space
    enqueue(&q, 70);
```

```
    display(&q);

    return 0;
}
```

10. Write a C program to implement Doubly Linked List.

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

// Define a node structure for the doubly linked list
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = newNode->prev = NULL;
```

```
    return newNode;
}

// Function to insert a node at the end of the list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

// Function to display the list from the beginning
void displayForward(struct Node* head) {
    struct Node* temp = head;
    if (temp == NULL) {
```

```
        printf("List is empty.\n");
        return;
    }
    printf("Doubly Linked List (Forward): ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Function to display the list from the end
void displayBackward(struct Node* head) {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("List is empty.\n");
        return;
    }
    // Move to the last node
    while (temp->next != NULL) {
        temp = temp->next;
    }
```

```
    printf("Doubly Linked List (Backward): ");
    // Traverse backward using the prev pointer
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->prev;
    }
    printf("\n");
}

// Function to delete a node with a given value
void deleteNode(struct Node** head, int data) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = *head;
    // Search for the node to delete
    while (temp != NULL && temp->data != data) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Node with data %d not found.\n", data);
```

```
        return;
    }
    // If the node to be deleted is the head node
    if (temp == *head) {
        *head = temp->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(temp);
        printf("Node with data %d deleted.\n", data);
        return;
    }
    // If the node to be deleted is not the head
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }
    free(temp);
    printf("Node with data %d deleted.\n", data);
}
```

```
// Function to free the entire list
void freeList(struct Node** head) {
    struct Node* temp = *head;
    while (temp != NULL) {
        struct Node* nextNode = temp->next;
        free(temp);
        temp = nextNode;
    }
    *head = NULL;
}

int main() {
    struct Node* head = NULL;

    // Inserting nodes into the doubly linked list
    insertAtEnd(&head, 10);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    insertAtEnd(&head, 40);
    insertAtEnd(&head, 50);
```

```
    // Display the list forward and backward
    displayForward(head);
    displayBackward(head);

    // Deleting a node with value 30
    deleteNode(&head, 30);

    // Display the list again after deletion
    displayForward(head);
    displayBackward(head);

    // Free the entire list
    freeList(&head);

    return 0;
}
```