I N D E X

Experiment No: 1:

Problem Statement: Write a program to add and multiply two matrixes.

A) Write a programme to add two matrices.

```
//C Programme No:  to add and multiply two compatible matrices
#include <bits/stdc++.h>
using namespace std;
// Programme to Show Matrics addition
int main(){
    int n, m, o, p;
    cout << "Enter Row of Matrix 1: ";
    cin >> n;
    cout << "Enter Column of Matrix 1: ";
    cin >> m;

    cout << "Enter Row of Matrix 2: ";
    cin >> o;
    cout << "Enter Column of Matrix 2: ";
    cin >> p;

    int arr1[n][m], arr2[o][p], arr3[10][10] = {0};
    cout << "Enter elements of Matrix 1:" << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cin >> arr1[i][j];
        }
    }

    cout << "Enter elements of Matrix 2:" << endl;
    for (int i = 0; i < o; i++)
    {
        for (int j = 0; j < p; j++)
        {
            cin >> arr2[i][j];
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < p; j++)
        {
            for (int k = 0; k < o; k++)
            {
                arr3[i][j] += arr1[i][k] + arr2[k][j];
            }
```

```
            }
        }

        cout << "\nAddition of Two Matrices:" << endl;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < p; j++)
            {
                cout << arr3[i][j] << " ";
            }
            cout << endl;
        }
        return 0;
}
```

OUTPUT: -

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Programmes\2D Array> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Programmes\2D Array\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Programmes\2D Array\output> & .\'arrayAddition.exe'
Enter Row of Matrix 1: 3
Enter Column of Matrix 1: 3
Enter Row of Matrix 2: 3
Enter Column of Matrix 2: 3
Enter elements of Matrix 1:
1 2 3
4 5 6
7 8 9
Enter elements of Matrix 2:
9 8 7
6 5 4
3 2 1

Addition of Two Matrices:
24 21 18
33 30 27
42 39 36
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Programmes\2D Array\output>
```

B) Write a programme to multiply two matrices.

```
#include <bits/stdc++.h>
using namespace std;
// Program to show Matrics Multiplication
int main() {
    int n, m, o, p;
    cout << "Enter Row of Matrix 1: ";
    cin >> n;
    cout << "Enter Column of Matrix 1: ";
    cin >> m;

    cout << "Enter Row of Matrix 2: ";
    cin >> o;
    cout << "Enter Column of Matrix 2: ";
    cin >> p;
    if (m == o) {
        int arr1[n][m], arr2[o][p], arr3[10][10] = {0};
        cout << "Enter elements of Matrix 1:" << endl;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                cin >> arr1[i][j];
            }
        }
        cout << "Enter elements of Matrix 2:" << endl;
        for (int i = 0; i < o; i++) {
            for (int j = 0; j < p; j++) {
                cin >> arr2[i][j];
            }
        }
        // Matrix Multiplication
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < p; j++) {
                for (int k = 0; k < o; k++) {
                    arr3[i][j] += arr1[i][k] * arr2[k][j];
                }
            }
        }
        cout << "\nMultiplication of Two Matrices:" << endl;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < p; j++) {
                cout << arr3[i][j] << " ";
            }
            cout << endl;
        }
    } else {
        cout << "Matrix multiplication is not possible." <<
endl;
    }
    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Programmes\2D Array\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Programmes\2D Array\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Programmes\2D Array\output> & .\'arrayrMul.exe'
Enter Row of Matrix 1: 3
Enter Column of Matrix 1: 3
Enter ROw of Matrix 2: 3
Enter Column of Matrix 2: 3
Enter elements of Matrix 1:
1 2 3
4 5 6
7 8 9
Enter elements of Matrix 2:
9 8 7
6 5 4
3 2 1

Multiplication of Two Matrices:
30 24 18
84 69 54
138 114 90
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Programmes\2D Array\output>
```

Experiment No: 2

Problem Statement: Write a program to Transpose the matrix.

```c
#include <stdio.h>
#include <conio.h>
#define N 50

 int main(){
 int m, n, i, j, matrix[N][N], transpose[N][N];

 printf("Enter rows and columns : ");
 scanf("%d%d", &m, &n);

 printf("Enter elements of the matrix : \n");
 for (i= 0; i < m; i++)
     for (j = 0; j < n; j++)
         scanf("%d", &matrix[i][j]);

 for (i = 0;i < m;i++)
     for (j = 0; j < n; j++)
         transpose[j][i] = matrix[i][j];

 printf("Transpose of the matrix:\n");
 for (i = 0; i< n; i++) {
     for (j = 0; j < m; j++)
         printf("%d  ", transpose[i][j]);
     printf("\n");
 }

 return 0;
}
```

OUTPUT: -

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q2.exe'
Enter rows and columns : 3 3
Enter elements of the matrix :
1 2 3
4 5 6
7 8 9
Transpose of the matrix:
1 4 7
2 5 8
3 6 9
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 3

Problem Statement: Write a program to Implement Stack Using Array.

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

#define MAX_SIZE 5

typedef struct {
    int stackArray[MAX_SIZE];
    int top;
}Mystack;

void init_stack(Mystack *stack){
    stack->top = -1;
}

void push(Mystack *stack, int value){
    if(stack->top == MAX_SIZE-1){
        printf("Stack is full you cannot push element\n");
        return;
    }
    stack->stackArray[++stack->top] = value;
}

int pop(Mystack *stack){
    if(stack->top == -1){
        printf("Stack is empty\n");
        return -1;
    }
    return stack->stackArray[stack->top--];
}

int peek(Mystack *stack){
    if(stack->top == -1){
        printf("Stack is empty\n");
        return -1;
    }
    return stack->stackArray[stack->top];
}

bool isEmpty(Mystack *stack){
    return (stack->top == -1);
}

bool isfull(Mystack *stack){
```

```
    return (stack->top == MAX_SIZE - 1);
}

int main() {
    Mystack stack;
    init_stack(&stack);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);

    printf("peek : %d\n", peek(&stack));

    printf("Pop : %d\n", pop(&stack));
    printf("Pop : %d\n", pop(&stack));

    push(&stack, 40);
    push(&stack, 50);
    push(&stack, 60);
    push(&stack, 70);
    push(&stack, 80); // here our stack is full

    printf("peek : %d\n", peek(&stack));

}
```

OUTPUT: -

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q4.exe'
10 pushed to stack.
20 pushed to stack.
30 pushed to stack.
Top element is 30.
30 popped from stack.
20 popped from stack.
Top element is 10.
40 pushed to stack.
Top element is 40.
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 4

Problem Statement: Write a program to Implement Stack Using Linked List.

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

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

// Define a structure for the stack
struct Stack {
    struct Node* top;
};

// 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 = NULL;
    return newNode;
}

// Function to initialize a stack
struct Stack* initializeStack() {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct
Stack));
    stack->top = NULL;
    return stack;
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
    return (stack->top == NULL);
}

// Function to push an element onto the stack
void push(struct Stack* stack, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = stack->top;
    stack->top = newNode;
    printf("%d pushed to stack.\n", data);
}
```

```c
// Function to pop an element from the stack
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow.\n");
        return -1;
    }
    struct Node* temp = stack->top;
    int popped = temp->data;
    stack->top = stack->top->next;
    free(temp);
    return popped;
}

// Function to peek at the top element of the stack
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
        return -1;
    }
    return stack->top->data;
}

// Main function to test the stack implementation
int main() {
    struct Stack* stack = initializeStack();
    push(stack, 10);
    push(stack, 20);
  push(stack, 30);

    printf("Top element is %d.\n", peek(stack));
    printf("%d popped from stack.\n", pop(stack));
    printf("%d popped from stack.\n", pop(stack))
    printf("Top element is %d.\n", peek(stack));

    push(stack, 40);

    printf("Top element is %d.\n", peek(stack));

    return 0;
}
```

OUTPUT: -

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q4.exe'
10 pushed to stack.
20 pushed to stack.
30 pushed to stack.
Top element is 30.
30 popped from stack.
20 popped from stack.
Top element is 10.
40 pushed to stack.
Top element is 40.
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 5

Problem Statement: Write a program to Implement queue Using Array.

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

// Structure for stack node
struct StackNode {
    int data;
    struct StackNode* next;
};

// Function to create a new stack node
struct StackNode* newNode(int data) {
    struct StackNode* stackNode = (struct
StackNode*)malloc(sizeof(struct StackNode));
    stackNode->data = data;
    stackNode->next = NULL;
    return stackNode;
}

// Function to push an element onto the stack
void push(struct StackNode** top, int data) {
    struct StackNode* stackNode = newNode(data);
    stackNode->next = *top;
    *top = stackNode;
}

// Function to pop an element from the stack
int pop(struct StackNode** top) {
    if (*top == NULL) {
        printf("Stack underflow\n");
        return -1;
    }
    struct StackNode* temp = *top;
    *top = (*top)->next;
    int popped = temp->data;
    free(temp);
    return popped;
}

// Structure for queue using two stacks
struct Queue {
    struct StackNode* stack1;
    struct StackNode* stack2;
};
```

```c
// Function to enqueue an element into the queue
void enqueue(struct Queue* queue, int data) {
    push(&queue->stack1, data);
}

// Function to dequeue an element from the queue
int dequeue(struct Queue* queue) {
    int item;
    if (queue->stack1 == NULL && queue->stack2 == NULL) {
        printf("Queue is empty\n");
        return -1;
    }
    if (queue->stack2 == NULL) {
        while (queue->stack1 != NULL) {
            item = pop(&queue->stack1);
            push(&queue->stack2, item);
        }
    }
    item = pop(&queue->stack2);
    return item;
}

int main() {
    // Create a queue
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct
Queue));
    queue->stack1 = NULL;
    queue->stack2 = NULL;

    // Enqueue elements
    enqueue(queue, 1);
    enqueue(queue, 2);
    enqueue(queue, 3);

    // Dequeue and print elements
    printf("%d\n", dequeue(queue));
    printf("%d\n", dequeue(queue));
    printf("%d\n", dequeue(queue));

    free(queue); // free allocated memory
    return 0;
}
```

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\out
put'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q6.exe'
1
2
3
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 6

Problem Statement: Write a program to Implement circular queue Using Array.

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

#define MAX_SIZE 5 // Maximum size of the circular queue

// Structure for the circular queue
struct CircularQueue {
    int* array; // Array to store elements
    int front; // Front of the queue
    int rear; // Rear of the queue
    int size; // Current size of the queue
};

// Function to create a circular queue
struct CircularQueue* createCircularQueue() {
    struct CircularQueue* cq = (struct
CircularQueue*)malloc(sizeof(struct CircularQueue));
    cq->array = (int*)malloc(MAX_SIZE * sizeof(int));
    cq->front = -1;
    cq->rear = -1;
    cq->size = 0;
    return cq;
}

// Function to check if the circular queue is full
int isFull(struct CircularQueue* cq) {
    return (cq->front == (cq->rear + 1) % MAX_SIZE && cq->size
== MAX_SIZE);
}

// Function to check if the circular queue is empty
int isEmpty(struct CircularQueue* cq) {
    return (cq->front == -1 && cq->rear == -1 && cq->size ==
0);
}

// Function to enqueue an element into the circular queue
void enqueue(struct CircularQueue* cq, int data) {
    if (isFull(cq)) {
        printf("Queue is full\n");
        return;
    } else {
        cq->rear = (cq->rear + 1) % MAX_SIZE;
```

```
        cq->array[cq->rear] = data;
        if (cq->front == -1)
            cq->front = cq->rear;
        cq->size++;
        printf("%d enqueued to the queue\n", data);
    }
}

// Function to dequeue an element from the circular queue
int dequeue(struct CircularQueue* cq) {
    int data;
    if (isEmpty(cq)) {
        printf("Queue is empty\n");
        return -1;
    } else {
        data = cq->array[cq->front];
        if (cq->front == cq->rear) {
            cq->front = -1;
            cq->rear = -1;
        } else {
            cq->front = (cq->front + 1) % MAX_SIZE;
        }
        cq->size--;
        return data;
    }
}

// Function to display the circular queue
void display(struct CircularQueue* cq) {
    int i;
    if (isEmpty(cq)) {
        printf("Queue is empty\n");
        return;
    }
    printf("Elements in Circular Queue are: ");
    for (i = cq->front; i != cq->rear; i = (i + 1) % MAX_SIZE)
        printf("%d ", cq->array[i]);
    printf("%d\n", cq->array[i]);
}

int main() {
    struct CircularQueue* cq = createCircularQueue();

    // Enqueue elements
    enqueue(cq, 1);
    enqueue(cq, 2);
    enqueue(cq, 3);
    enqueue(cq, 4);
```

```
    enqueue(cq, 5);

    // Display elements
    display(cq);

    // Dequeue elements
    printf("Dequeued element: %d\n", dequeue(cq));
    printf("Dequeued element: %d\n", dequeue(cq));

    // Display elements
    display(cq);

    // Enqueue elements
    enqueue(cq, 6);
    enqueue(cq, 7);

    // Display elements
    display(cq);

    free(cq->array); // free allocated memory for array
    free(cq); // free allocated memory for circular queue
    return 0;
}
```

Output

Experiment No: 7

Problem Statement: Write a program to Implement queue Using Linked List.

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

// Structure for a node in the queue
struct Node {
    int data;
    struct Node* next;
};

// Structure for the queue
struct Queue {
    struct Node* front; // Front of the queue
    struct Node* rear; // Rear of the queue
};

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

// Function to create an empty queue
struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct
Queue));
    q->front = NULL;
    q->rear = NULL;
    return q;
}

// Function to check if the queue is empty
int isEmpty(struct Queue* q) {
    return (q->front == NULL);
}

// Function to enqueue an element into the queue
void enqueue(struct Queue* q, int data) {
    struct Node* temp = newNode(data);
    if (isEmpty(q)) {
        q->front = temp;
```

```
    } else {
        q->rear->next = temp;
    }
    q->rear = temp;
    printf("%d enqueued to the queue\n", data);
}

// Function to dequeue an element from the queue
int dequeue(struct Queue* q) {
    int data;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    struct Node* temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) // If front becomes NULL, set rear
to NULL as well
        q->rear = NULL;
    data = temp->data;
    free(temp);
    return data;
}

// Function to display the queue
void display(struct Queue* q) {
    struct Node* temp = q->front;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    printf("Elements in Queue are: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Queue* q = createQueue();

    // Enqueue elements
    enqueue(q, 1);
    enqueue(q, 2);
    enqueue(q, 3);
    enqueue(q, 4);
    enqueue(q, 5);
```

```c
    // Display elements
    display(q);

    // Dequeue elements
    printf("Dequeued element: %d\n", dequeue(q));
    printf("Dequeued element: %d\n", dequeue(q));

    // Display elements
    display(q);

    // Enqueue elements
    enqueue(q, 6);
    enqueue(q, 7);

    // Display elements
    display(q);

    free(q); // free allocated memory for queue
    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q8.exe'
1 enqueued to the queue
2 enqueued to the queue
3 enqueued to the queue
4 enqueued to the queue
5 enqueued to the queue
Elements in Queue are: 1 2 3 4 5
Dequeued element: 1
Dequeued element: 2
Elements in Queue are: 3 4 5
6 enqueued to the queue
7 enqueued to the queue
Elements in Queue are: 3 4 5 6 7
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 8

Problem Statement: Write a program to Implement BFS Using Linked List.

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

// Structure for a node in the graph
struct Node {
    int data;
    struct Node* next;
};

// Structure for a graph
struct Graph {
    int numVertices;
    struct Node** adjLists;
    bool* visited;
};

// 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 = NULL;
    return newNode;
}

// Function to create a graph with a given number of vertices
struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct
Graph));
    graph->numVertices = vertices;
    graph->adjLists = (struct Node**)malloc(vertices *
sizeof(struct Node*));
    graph->visited = (bool*)malloc(vertices * sizeof(bool));

    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = false;
    }

    return graph;
}
```

```c
// Function to add an edge between two vertices
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function to perform BFS traversal starting from a given
vertex
void BFS(struct Graph* graph, int startVertex) {
    // Create a queue for BFS
    int queue[graph->numVertices];
    int front = -1, rear = -1;

    // Mark the current vertex as visited and enqueue it
    graph->visited[startVertex] = true;
    queue[++rear] = startVertex;

    while (front != rear) {
        // Dequeue a vertex from queue and print it
        int currentVertex = queue[++front];
        printf("%d ", currentVertex);

        // Get all adjacent vertices of the dequeued vertex
currentVertex
        struct Node* temp = graph->adjLists[currentVertex];

        while (temp) {
            int adjVertex = temp->data;
            if (!graph->visited[adjVertex]) {
                graph->visited[adjVertex] = true;
                queue[++rear] = adjVertex;
            }
            temp = temp->next;
        }
    }
}

// Function to display the adjacency list representation of
the graph
void printGraph(struct Graph* graph) {
    for (int i = 0; i < graph->numVertices; i++) {
```

```
        struct Node* temp = graph->adjLists[i];
        printf("Vertex %d: ", i);
        while (temp) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int main() {
    int numVertices = 5;
    struct Graph* graph = createGraph(numVertices);

    // Add edges
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);

    printf("Adjacency list representation of the graph:\n");
    printGraph(graph);

    printf("\nBFS traversal starting from vertex 0: ");
    BFS(graph, 0);

    // Free allocated memory
    for (int i = 0; i < numVertices; i++) {
        struct Node* temp = graph->adjLists[i];
        while (temp) {
            struct Node* prev = temp;
            temp = temp->next;
            free(prev);
        }
    }
    free(graph->adjLists);
    free(graph->visited);
    free(graph);

    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q9.exe'
Adjacency list representation of the graph:
Vertex 0: 2 -> 1 -> NULL
Vertex 1: 4 -> 3 -> 0 -> NULL
Vertex 2: 0 -> NULL
Vertex 3: 1 -> NULL
Vertex 4: 1 -> NULL

BFS traversal starting from vertex 0: 0 2 1 4 3
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 9

Problem Statement: Write a program to Implement DFS Using Linked List.

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

// Structure for a node in the graph
struct Node {
    int data;
    struct Node* next;
};

// Structure for a graph
struct Graph {
    int numVertices;
    struct Node** adjLists;
    bool* visited;
};

// 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 = NULL;
    return newNode;
}

// Function to create a graph with a given number of vertices
struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct
Graph));
    graph->numVertices = vertices;
    graph->adjLists = (struct Node**)malloc(vertices *
sizeof(struct Node*));
    graph->visited = (bool*)malloc(vertices * sizeof(bool));

    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = false;
    }

    return graph;
}
```

```
// Function to add an edge between two vertices
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function for DFS traversal starting from a given vertex
void DFS(struct Graph* graph, int vertex) {
    // Mark the current vertex as visited
    graph->visited[vertex] = true;
    printf("%d ", vertex);

    // Get all adjacent vertices of the current vertex
    struct Node* temp = graph->adjLists[vertex];

    while (temp != NULL) {
        int adjVertex = temp->data;
        if (!graph->visited[adjVertex]) {
            DFS(graph, adjVertex); // Recursive call for
unvisited adjacent vertex
        }
        temp = temp->next;
    }
}

// Function to display the adjacency list representation of
the graph
void printGraph(struct Graph* graph) {
    for (int i = 0; i < graph->numVertices; i++) {
        struct Node* temp = graph->adjLists[i];
        printf("Vertex %d: ", i);
        while (temp) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int main() {
    int numVertices = 5;
```

```c
    struct Graph* graph = createGraph(numVertices);

    // Add edges
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);

    printf("Adjacency list representation of the graph:\n");
    printGraph(graph);

    printf("\nDFS traversal starting from vertex 0: ");
    DFS(graph, 0);

    // Free allocated memory
    for (int i = 0; i < numVertices; i++) {
        struct Node* temp = graph->adjLists[i];
        while (temp) {
            struct Node* prev = temp;
            temp = temp->next;
            free(prev);
        }
    }
    free(graph->adjLists);
    free(graph->visited);
    free(graph);

    return 0;
}
```

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q10.exe'
Adjacency list representation of the graph:
Vertex 0: 2 -> 1 -> NULL
Vertex 1: 4 -> 3 -> 0 -> NULL
Vertex 2: 0 -> NULL
Vertex 3: 1 -> NULL
Vertex 4: 1 -> NULL

DFS traversal starting from vertex 0: 0 2 1 4 3
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 10

Problem Statement: Write a program to Implement Linear Search.

```c
#include <stdio.h>

// Function to perform linear search
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i; // Return the index of the key if found
        }
    }
    return -1; // Return -1 if key is not found
}

int main() {
    int arr[] = {2, 5, 7, 9, 11, 13, 17};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 11;

    int index = linearSearch(arr, n, key);

    if (index != -1) {
        printf("Element %d found at index %d.\n", key, index);
    } else {
        printf("Element %d not found in the array.\n", key);
    }

    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q11.exe'
Element 11 found at index 4.
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 11

Problem Statement: Write a program to Implement Binary Search.

```c
#include <stdio.h>
// Function to perform binary search
int binarySearch(int arr[], int low, int high, int key) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        // Check if the key is present at mid
        if (arr[mid] == key) {
            return mid;
        }
        // If key is greater, ignore left half
        if (arr[mid] < key) {
            low = mid + 1;
        }
        // If key is smaller, ignore right half
        else {
            high = mid - 1;
        }
    }
    // Key not found, return -1
    return -1;
}
int main() {
    int arr[] = {2, 5, 7, 9, 11, 13, 17};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 11;
    int index = binarySearch(arr, 0, n - 1, key);
    if (index != -1) {
        printf("Element %d found at index %d.\n", key, index);
    } else {
        printf("Element %d not found in the array.\n", key);
    }
    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q12.exe'
Element 11 found at index 4.
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 12

Problem Statement: Write a program to Implement Bubble Sorting.

```c
#include <stdio.h>

// Function to perform bubble sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        // Flag to check if any swapping is done in this pass
        int swapped = 0;

        // Last i elements are already in place, so we don't
need to compare them
        for (int j = 0; j < n - i - 1; j++) {
            // If current element is greater than the next
element, swap them
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = 1; // Set the flag to indicate
swapping is done
            }
        }

        // If no swapping is done in this pass, array is
already sorted
        if (swapped == 0) {
            break;
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    bubbleSort(arr, n);

    printf("Sorted array: ");
```

```
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\
Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q13.exe'
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 13

Problem Statement: Write a program to Implement Selection Sorting.

```c
#include <stdio.h>
// Function to perform selection sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        // Find the index of the minimum element in the
unsorted part of the array
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swap the minimum element with the first element of
the unsorted part
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}
int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    selectionSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q14.exe'
Original array: 64 25 12 22 11
Sorted array: 11 12 22 25 64
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 14

Problem Statement: Write a program to Implement Insertion Sorting.

```c
#include <stdio.h>
// Function to perform insertion sort
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements of arr[0..i-1], that are greater than
key,
        // to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    insertionSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\
Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q15.exe'
Original array: 12 11 13 5 6
Sorted array: 5 6 11 12 13
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 15

Problem Statement: Write a program to Implement Merge Sorting.

```c
#include <stdio.h>
#include <stdlib.h>
// Merge two subarrays of arr[]
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    // Create temporary arrays
    int L[n1], R[n2];
    // Copy data to temporary arrays L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    // Merge the temporary arrays back into arr[l..r]
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    // Copy the remaining elements of L[], if any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    // Copy the remaining elements of R[], if any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

```c
// l is for left index and r is right index of the sub-array
of arr to be sorted
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for large l
and r
        int m = l + (r - l) / 2;
        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}
// Utility function to print an array
void printArray(int A[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);
    printf("Given array is \n");
    printArray(arr, arr_size);
    mergeSort(arr, 0, arr_size - 1);
    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\
Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q16.exe'
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 16

Problem Statement: Write a program to Implement Heap Sorting.

```c
#include <stdio.h>
// Function to heapify a subtree rooted with node i, which is
an index in arr[]
void heapify(int arr[], int n, int i) {
    int largest = i;    // Initialize largest as root
    int l = 2 * i + 1;  // Left child
    int r = 2 * i + 2;  // Right child

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        // Swap arr[i] and arr[largest]
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// Main function to do heap sort
void heapSort(int arr[], int n) {
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        // Call max heapify on the reduced heap
        heapify(arr, i, 0);
```

```
    }
}

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

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, n);

    heapSort(arr, n);

    printf("Sorted array is \n");
    printArray(arr, n);
    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> cd 'd:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output> & .\'Q17.exe'
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos\output>
```

Experiment No: 17

Problem Statement: Write a program to Implement Matrix Multiplication by Strassen's algorithm.

```
#include<stdio.h>
int main(){
  int a[2][2], b[2][2], c[2][2], i, j;
  int m1, m2, m3, m4 , m5, m6, m7;

  printf("Enter the 4 elements of first matrix: ");
  for(i = 0;i < 2; i++)
      for(j = 0;j < 2; j++)
           scanf("%d", &a[i][j]);

  printf("Enter the 4 elements of second matrix: ");
  for(i = 0; i < 2; i++)
      for(j = 0;j < 2; j++)
           scanf("%d", &b[i][j]);

  printf("\nThe first matrix is\n");
  for(i = 0; i < 2; i++){
      printf("\n");
      for(j = 0; j < 2; j++)
           printf("%d\t", a[i][j]);
  }
  printf("\nThe second matrix is\n");
  for(i = 0;i < 2; i++){
      printf("\n");
      for(j = 0;j < 2; j++)
           printf("%d\t", b[i][j]);
  }
  m1= (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
  m2= (a[1][0] + a[1][1]) * b[0][0];
```

```
  m3= a[0][0] * (b[0][1] - b[1][1]);
  m4= a[1][1] * (b[1][0] - b[0][0]);
  m5= (a[0][0] + a[0][1]) * b[1][1];
  m6= (a[1][0] - a[0][0]) * (b[0][0]+b[0][1]);
  m7= (a[0][1] - a[1][1]) * (b[1][0]+b[1][1]);
  c[0][0] = m1 + m4- m5 + m7;
  c[0][1] = m3 + m5;
  c[1][0] = m2 + m4;
  c[1][1] = m1 - m2 + m3 + m6;
   printf("\nAfter multiplication using Strassen's algorithm
\n");
   for(i = 0; i < 2 ; i++){
      printf("\n");
      for(j = 0;j < 2; j++)
           printf("%d\t", c[i][j]);
   }
   return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos> ./Q18.exe
Enter the 4 elements of first matrix: 1 2 3 4
Enter the 4 elements of second matrix: 1 2 3 4

The first matrix is

1       2
3       4
The second matrix is

1       2
3       4
After multiplication using Strassen's algorithm

7       10
15      22
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos>
```

Experiment No: 18

Problem Statement: Write a program to Find Minimum Spanning Tree Using Kruskal's Algorithm.

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

// Structure to represent an edge in the graph
struct Edge {
    int src, dest, weight;
};

// Structure to represent a subset for union-find
struct Subset {
    int parent;
    int rank;
};

// Function to find the set of an element 'i' (uses path
compression technique)
int find(struct Subset subsets[], int i) {
    // Find root and make root as parent of i (path
compression)
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

// Function to do union of two sets of x and y (uses union by
rank)
void Union(struct Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Attach smaller rank tree under root of high rank tree
(Union by Rank)
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
    else {
        // If ranks are the same, then make one as root and
increment its rank by one
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
```

```
}

// Compare two edges according to their weights
int compare(const void* a, const void* b) {
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight - b1->weight;
}

// Function to construct MST using Kruskal's algorithm
void KruskalMST(struct Edge edges[], int V, int E) {
    struct Edge result[V]; // To store the resultant MST
    int e = 0; // Index variable used for result[]
    int i = 0; // Index variable used for sorted edges[]

    // Step 1: Sort all the edges in non-decreasing order of
their weight
    qsort(edges, E, sizeof(edges[0]), compare);

    // Allocate memory for creating V subsets
    struct Subset* subsets = (struct Subset*)malloc(V *
sizeof(struct Subset));

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Number of edges to be taken is equal to V-1
    while (e < V - 1 && i < E) {
        // Step 2: Pick the smallest edge. And increment the
index for the next iteration
        struct Edge next_edge = edges[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        // If including this edge does't cause cycle, include
it in result and increment the index of result for next edge
        if (x != y) {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
        // Else discard the next_edge
    }

    // Print the contents of result[] to display the built MST
```

```
    printf("Following are the edges in the constructed MST\n");
    for (i = 0; i < e; ++i)
        printf("%d -- %d == %d\n", result[i].src,
result[i].dest, result[i].weight);
}

int main() {
    int V = 4; // Number of vertices in the graph
    int E = 5; // Number of edges in the graph
    struct Edge edges[] = {
        {0, 1, 10}, {0, 2, 6}, {0, 3, 5}, {1, 3, 15}, {2, 3, 4}
    };

    KruskalMST(edges, V, E);

    return 0;
}
```

Output

```
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos>  & 'c:\Users\pappu\.vscode\extensions\ms-vscode.cpptools-1.19.9-win32-x64
\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-qy4j1115.4r1' '--stdout=Microsoft-MIEngine-Out-ayak5u0e.1gy' '--std
err=Microsoft-MIEngine-Error-5fsz4nnk.h3w' '--pid=Microsoft-MIEngine-Pid-pksznjhu.uof' '--dbgExe=C:\MinGW\bin\gdb.exe' '--interpreter=mi'
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos> ./Q19.exe
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
PS D:\Galgotias MCA Study\MCA\Semester notes\2nd Sem\DSA Using C\Cos>
```